(4)

RADC-TR-87-192
Final Technical Report
November 1987

AD-A193 145

THEORY OF ENDORSEMENTS AND REASONING WITH UNCERTAINTY, JANUARY 1984 - JANUARY 1986

University of Massachusetts

Sponsored by
Defense Advanced Research Projects Agency (DOD)
ARPA Order No. 5294

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-87-192 has been reviewed and is approved for publication.

APPROVED: Chuian-Chuian Hwong

CHUIAN-CHUIAN HWONG
Project Engineer

APPROVED: Raymond P. Urtz Jr.

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: James W. Hyde III

JAMES W. HYDE III.
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notice on a specific document requires that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-87-192
6a. NAME OF PERFORMING ORGANIZATION University of Massachusetts 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COES)
6c. ADDRESS (City, State, and ZIP Code) Dept of Computer and Information Science Lederle Graduate Research Center Amherst MA 01003	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Defense Advanced Research Projects Agency 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0014
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Boulevard Arlington VA 22209-2308	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62301E; PROJECT NO. E294; TASK NO 00; WORK UNIT ACCESSION NO. 01
11. TITLE (Include Security Classification) THEORY OF ENDORSEMENT AND REASONING WITH UNCERTAINTY, JANUARY 1984 - JANUARY 1986	
12. PERSONAL AUTHOR(S) Paul R. Cohen, Alvah David, David Day, Jeff Delisio, Mike Greenberg and Thomas Gruber	
13a. TYPE OF REPORT Final	13b. TIME COVERED FROM Jan 84 TO Jan 86
14. DATE OF REPORT (Year, Month, Day) November 1987	15. PAGE COUNT 166
16. SUPPLEMENTARY NOTATION N/A	

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
12	05	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Endorsement-Based Reasoning; Automated Knowledge Acquisition; Prospective Reasoning; Decision Making; Plausible Inference; Knowledge Engineering

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

By emphasizing the sources of uncertainty and its consequences, an intelligent reasoning system shows the ability to plan a course of action appropriate to one's uncertainty, the ability to explain one's actions, and the ability to determine degree of belief in alternatives given evidence. Other than numerical inferences and reason maintenance, another approach to parallel certainty inference, the theory of endorsements, is presented. The fundamental assumption of the theory of endorsements is that subjective degrees of belief are composites of reasons to believe and disbelieve (positive and negative endorsements). Two implementations of the theory of endorsements, SOLOMON and HMMM, are briefly described.

GRANT, a knowledge system that finds sources of funding for research proposals, was developed to explore the utility of semantic matching in uncertain domains. The basic mode of inference used in GRANT is plausible inference.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Chuian-Chuian Hwong	22b. TELEPHONE (Include Area Code) (315) 330-7794 22c. OFFICE SYMBOL RADC (COES)

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

UNCLASSIFIED

MU (Management of Uncertainty), a development environment which supports prospective reasoning, was used to reimplement a medical expert system, called MUM. MUM manages uncertainty by reasoning about evidence and its current state of belief in hypotheses.

A prospective, constructive decision making system, called CDM, was developed to define and evaluate decisions autonomously. Contrast with the more static decision theoretic models, CDM is dynamic in that the algorithm gathers evidence as the decision evolves at the most opportune time.

Three principles of design for knowledge acquisition are described and demonstrated their application in an architecture where knowledge about evidential combination and knowledge about control can be acquired from an expert. Proper design of knowledge representation primitives can reduce the representational mismatch between implementation-level and task-level formalisms.

This report is compiled from work by the following authors:

Paul R. Cohen
Alvah Davis
David Day
Jeff Delisio
Mike Greenberg
Thomas Gruber
Adele Howe
Rick Kjeldsen
Susan Lander
Cynthia Loiselle
Philip M. Stanhope
Michael Sullivan
Dan Suthers

Contents

Chapter 1

Endorsement-based Reasoning

1 Introduction

Uncertainty is a state of mind that arises in the reasoning process. Our approach is to ask what aspects of the process give rise to uncertainty. We emphasize the sources of uncertainty and its consequences, rather than uncertainty as a mental phenomenon. A second emphasis is how, in light of these sources and consequences, a system responds to uncertainty. When a system cannot change its behavior in response to uncertainty, which nonetheless has deleterious effects, we say it is reasoning *under* uncertainty. A system that incorporates in its problem-solving repertoire some kind of response to uncertainty is said to reason *with* uncertainty. A system that explicitly represents sources and consequences of uncertainty, and reasons about them to control its own behavior (e.g., by selecting problem-solving responses) is said to reason *about* uncertainty. Reasoning about uncertainty thus places the most responsibility for managing uncertainty on the system; reasoning with uncertainty is inflexible, because the system does not reason autonomously about *how* to manage its uncertainty; reasoning under uncertainty does not involve any management of uncertainty, autonomous or otherwise.

Our emphasis on the many sources of uncertainty has led us to a position we call the *composite* view of uncertainty, contrasting with the *one-dimensional* view. Consider a property of animals called "nastiness". We propose to rank animals on this one dimension by their nastiness: sharks and vipers are very, very nasty; shrews are a bit less nasty; and so on until we reach koala bears. The inquiring mind will look at our ranking and ask, "What features make one animal nastier than another?" because even though the ranking is on a single dimension, the features that contribute to nastiness are several. Those who must deal with nasty animals will want to know *why* their subjects are nasty – their nasty characteristics – not merely the extent of their nastiness. Just so with uncertainty. People and computers need to know why situations are uncertain, not merely the extent of their uncertainty. Thus, we believe that theories of uncertainty should emphasize the sources of uncertainty and its consequences.

2 Sources of Uncertainty

Uncertainty in rule-based inference has three general sources. It enters in *evidence*, which may be inaccurate or insufficient; it is implicit in any *model* of a domain (which is often encoded in production rules); and it is associated with the *beliefs* that result from inference. We discuss these in turn. Throughout the discussion we assume that the *environment* supplies evidence, which evokes *inferences*, which result in *beliefs*. Beliefs may be used as evidence for further inference.

2.1 Sources of Uncertainty About Evidence

Among the things we can say about evidence are that it is errorful, irrelevant, or insufficient. These are causes of uncertainty. In addition, we can say that a situation has a chance of being true; for example, it *might* rain, or Sally has an 80 percent chance of beating Fred's poker hand. Because we want to understand the sources of uncertainty, we are unwilling to summarize with a number the argument that evidence is, say, irrelevant; since we would no longer be able to distinguish irrelevance from insufficiency or other causes of uncertainty. We will try to maintain this distinction, though it is easily muddled when probabilistic arguments are combined with other causes of uncertainty; for example, the evidence "Sally's chances are 80 percent" may be insufficient, and evidence may have an 80 percent chance of being insufficient.

Errorful evidence is common in systems that rely on sensory information. For example, the tactile sensors of a robot may malfunction, resulting in an errorful interpretation of any evidence the sensor provides. Noise causes uncertainty about whether one's evidence is relevant. Systems such as HEARSAY-II (Erman, Lesser, Hayes-Roth, and Reddy, 1980) and HASP/SIAP (Nii and Feigenbaum, 1977) contended with noise from their transducers. Before they could ask whether data from transducers was errorful, they had first to cope with uncertainty about its relevance – whether it was signal or noise. Many applications are uncertain because they need more data than is readily available, quite apart from the questions of whether the data that *is* available is errorful or relevant. In medicine, for example, some invasive tests are expensive or life-threatening, and so diagnosis might proceed on the basis of incomplete evidence. In other cases, the needed evidence will never be available; for example, pollsters necessarily make statistical inferences from small samples because it is impossible, or impractical, to query an entire population.

We prefer to characterize a poll as "accurate within a 2 percent margin of error," or a diagnosis as "lacking the evidence from a brain scan," since these characterizations guide us in dealing with our uncertainty. The more we know about the causes and consequences of uncertainty about evidence, the better we are able to cope with the uncertainty.

2.2 Sources of Uncertainty About the Model

Rule-based inference systems capture knowledge about the world in inference rules (which constitute a *world model*). Uncertainty is caused by the processes of constructing and using these rules. When constructing rules, uncertainty is an inevitable consequence of summarizing knowledge. We recognize that expert inference rules are *compilations* of dozens or hundreds of experiences, and that minor differences between the experiences are "smoothed out" in the rule. When using such rules, "relevant" features of a case – those mentioned in the condition of a rule – are attended to, but discrepant features are ignored.

A related source of uncertainty in rule-based inference is that rules are constructed with some purpose in mind, but the context in which rules are used does not necessarily correspond to the purpose for which they were intended. For example, imagine a simple rule that states "If it's raining, then take an umbrella." This rule assumes that one's purpose is to stay dry, when in fact one may want to be drenched. It doesn't work to add another conditional clause to the rule, specifying that one wants to stay dry, because other implicit assumptions are easily generated; for example, we are assuming that the umbrella works. One cannot escape the uncertainty caused by not knowing whether the implicit assumptions of an inference rule are met.

Uncertainty arises from limitations of the world model. In terms of rule-based inference systems, uncertainty is caused by not knowing whether one has rules for all situations that may arise. It is worth making this source of uncertainty explicit, because it makes an interesting qualification on one's conclusions. Expert knowledge may be relatively complete, so when the expert says "As far as I know, you are healthy," you can be pleased. But the knowledge of expert systems is usually less complete, so a clean bill of health from one of *them* is probably less encouraging. The expert system *should* say "As far as I know, ...," because far from being a conversational nicety, it is an important qualification.

Note that "I'm pretty sure you're healthy" is not as informative as "As far as I know, you're healthy," since the latter states the cause of any uncertainty and the former does not. We re-emphasize the point we first made in connection with uncertainty about evidence: The more one knows about the sources of uncertainty about inference rules, the better one might cope with this uncertainty.

2.3 Sources of Uncertainty About Belief

Beliefs, in our simplified model of rule-based inference, are the conclusions of inferences. Thus, important sources of uncertainty in beliefs are uncertainty in evidence and inference rules. We will discuss how these sources of uncertainty are *propagated* to beliefs in later sections. Here, we concentrate on uncertainty that arises from one's beliefs independent of their derivation. The chief cause of uncertainty is that beliefs are sometimes *inconsistent*. For example, we believe that we pay too much money in taxes,

but we also believe that taxation for social programs is a Good Thing. Inconsistent beliefs lead to uncertainty about our future conclusions and actions; it is not possible to predict with certainty whether we will vote for tax-cutting or tax-raising political candidates. Another source of uncertainty concerns how beliefs are accessed. In humans, at least, one can easily demonstrate *priming* and *availability* biases (e.g., Kahneman, Slovic, and Tversky, 1982). Briefly, people do not bring all beliefs to mind with equal facility, and we use apparent facility as evidence about the truth of statements. In the simplest case, if we cannot think of any examples of a proposition (e.g., that books can dance the polka) then we say the proposition is false. This is fair enough, but we often misjudge the likelihood of propositions by this same device. In AI inference systems, access can be interpreted as search, which may be bounded by resource considerations, or deduction, also susceptible to limits. Since the structure of the representations of belief can affect the efficiency of access, judgments based on the relative ease of access can introduce uncertainty about beliefs regardless of their content.

3 Desiderata for Intelligent Reasoning About Uncertainty

This section asks what behaviors we should require of expert systems that reason intelligently about uncertainty. The requirements are of two kinds: first, we discuss what an expert system ought to *do* about uncertainty, then we focus on the representation of knowledge required to reason as we desire. It is striking that contemporary expert systems do very little about uncertainty besides measuring it. Some expert systems assess degrees of belief for hypotheses, but they do not use these numbers except to rank hypotheses and for some rudimentary control decisions. What more should an expert system do? We focus on two behaviors: planning (or control) and explanation.

Intelligent behavior under uncertainty requires a plan for the management of the uncertainty. Here are some examples of plans:

1. Confronted with uncertainty about which of two diseases afflict a patient, try to rule out the most serious one. Specifically, order relatively inexpensive, non-invasive tests before more costly ones, and give the patient a therapeutic trial of medication for the more serious disease. See the patient again after the test results are known and after the therapeutic trial has an opportunity to alleviate symptoms.

2. Since I am uncertain whether my weekday bus runs on the weekend, I decide to drive my car.

3. I am going to visit my parents, who say they have a birthday present for me. They won't tell me what it is, so just to be safe, I put the roof-rack on my car.

The first case is taken from a series of interviews with a physician on the problem of diagnosing chest pain. Two causes of pain, angina and esophageal spasm, can have identical manifestations, but one is more serious than the other. Thus, physicians will try to *rule out* angina first, and may prescribe therapy for angina on a trial basis. The angina/esophageal spasm differential is not usually resolved by ruling *in* esophageal spasm, since it is difficult to get direct, physical evidence of spasm. However, this plan is appropriate if less costly tests fail to resolve between the disease hypotheses. In contrast, one can sometimes quickly rule out angina by demonstrating that the pain is due to damage to the muscles of the chest. This "rule-out by ruling-in" plan may not be appropriate, however, if the patient is at risk for heart disease because of smoking, age, family history, and so on, since this patient may have *both* heart disease and some other cause of chest pain.

Thus, intelligent reasoning under uncertainty involves selecting a plan appropriate to the nature of the uncertainty. The "rule-out by ruling in" plan may be appropriate in some cases but not to the angina/esophageal spasm differential if the patient is at risk for heart disease and if less difficult tests have not yet been tried.

If one knows enough about the nature of one's uncertainty to intelligently select a plan, then this knowledge can be used to explain one's behavior:

- Why did you try to rule out angina before esophageal spasm?
- Because the consequences of my uncertainty about angina are more serious; and because it is difficult to find direct evidence for or against esophageal spasm; and because there is evidence that the patient is at risk for heart disease, so ruling in esophageal spasm would not rule out heart disease.

Many plans for managing uncertainty are much simpler than this one. The second example, above, is a case of sidestepping uncertainty. Instead of facing the uncertainty of whether a bus is running, the question is made irrelevant by deciding to drive a car. The third case is similar: it involves anticipating possible outcomes and preparing for the most extreme. When uncertain about the size of a birthday present, one prepares for the worst (best?) case by arranging transportation for the biggest possible object.

One characteristic of these examples is that the *probability* of the various uncertain outcomes is both insufficient to determine a response to the uncertainty, and furthermore, it is largely irrelevant. In the medical example, provided there is "enough" evidence for angina, the physician pursues the angina hypothesis not because it is more likely than esophageal spasm but because it is more dangerous. In the second case, if there is "not enough" evidence that the bus is running, the commuter decides to drive. The extent of the uncertainty in these cases, and the third case, is not the salient factor in deciding on a plan to manage the uncertainty.

Yet, the probability of outcomes plays a *small* role in these examples, and a greater role in other cases, such as this one:

An airplane has crashed in dense jungle. Searchers superimpose a grid on a map of the area and calculate, for each square in the grid, the probability that the plane crashed in that square. They search the high-probability areas first.
Here, the appropriate plan for managing uncertainty depends on knowing the likelihood of outcomes. Thus, in addition to planning and explanation, we need the ability to believe one proposition more than another. This, in turn, requires the ability to update degrees of belief in light of evidence.

In summary, the behaviors that make for intelligent reasoning about uncertainty are: the ability to plan a course of action appropriate to one's uncertainty, the ability to explain one's actions, and the ability to determine degrees of belief in alternatives given evidence. We now consider the conceptual tools required to build expert systems with these abilities.

An expert system requires a representation of knowledge about its uncertainty and methods for manipulating this knowledge to plan and explain actions, and to modify its belief in propositions. A good representation supports all the concepts one wishes to reason about, and all the methods one uses to reason about them. A good representation makes important distinctions explicit. One should not have to struggle to represent a situation — the representational techniques should make the "translation" between a situation and its representation easy. If these representational criteria are met, then we will be able to represent the knowledge required to achieve the three performance criteria outlined above. Table 1 summarizes the performance and representational criteria. We now survey current AI approaches to reasoning under uncertainty from the perspective of these criteria.

TABLE 1

Performance Criteria	
Planning:	Plan actions that are appropriate to uncertainty
Explanation:	Explain plans for managing uncertainty
Measurement:	Modify degree of belief in light of evidence
Representational criteria	
Adequacy:	Support all interesting concepts and methods for reasoning about them
Explicitness:	Make important distinctions explicit
Ease-of-use:	Make the "translation" between situation and representation easy

4 AI Approaches to Uncertainty

Given the diversity of sources of uncertainty, it is not surprising to find a plethora of responses in AI inference systems. The current approaches to uncertainty can be organized into three major groups. Systems constructed to circumvent the effects of uncertainty are of the *engineering approach*. Systems that control their behavior to avoid or reduce the effect of uncertainty use the *control approach*. Some systems divide the inference process into two separate subprocesses, one that performs inference as if there were no uncertainty, and another that associates representations of partial belief with the conclusions of the first process; this approach is called *parallel certainty inference* (Cohen, 1983). Although there is some overlap in these categories, they provide a useful organization to the discussion of current AI approaches to uncertainty.

4.1 The Engineering Approach

The designer of an inference system can anticipate some causes of uncertainty that effect the performance of a system, and then formulate the *problem* to eliminate any need to consider the uncertainty. For example, elementary textbook problems in physics ignore the effects of friction, relieving the student of the need to calculate the (uncertain) effect of this difficult-to-measure factor. It is common in AI inference systems to engineer the uncertainty out of problems, especially for prototype systems. Problems are frequently hard enough without considering noise or error, so the *clean data assumption* is often made to eliminate the effect of uncertainty introduced by the evidence. Of course, the same techniques that work with clean data must often be modified to cope with the problems of noise and error.

A second way to engineer uncertainty out of AI systems is to *assume relevance*. It is sometimes difficult to decide which features of the environment are relevant to a task, especially if one's world model is incomplete. Systems that are free of this uncertainty are conceptually simpler. For instance, Winston's (1975) "concept learning" program was presented with a set of training instances and inferred a "rule" to classify them. The program assumed that the teacher would supply typical and "near miss" cases of a special form. The problem was made tractable by assuming relevance, but the more difficult task of generating and evaluating training instances was finessed. Other learning systems (discussed in Dietterich, 1982; Michalski, Carbonell, and Mitchell, 1983) have made similar assumptions.

A third form of the engineering approach is a response to the kind of uncertainty that results from incomplete models of a domain. Since a system cannot know everything about its domain, it must make tentative decisions on the basis of uncertain beliefs. For instance, it is common to make the *closed world assumption* (Reiter, 1980) when working with a finite database of facts. The assumption asserts that something is false

if it is not known to be true.[1] Thus, under the closed world assumption, to decide whether X is true, one checks if X is known; if it is not known, then X is assumed to be false. In a rule-based inference system, if no rule has asserted a proposition (even though it is possible that one might in the future), the proposition is false under this assumption. A system that makes the closed world assumption is freed from the need to have a complete model; it has removed one source of uncertainty–the uncertainty of the unknown–by hiding it. (However, not all systems ignore the uncertainty introduced by assumptions. See the discussions of dependency-directed backtracking and reason maintenance in the following two sections for techniques that recognize and reason with the uncertainty introduced by assumptions.)

4.2 Control Approaches

Control approaches to uncertainty recognize the characteristics of a domain that cause uncertainty, and utilize control strategies to reduce the effects of uncertainty or eliminate its sources. As an illustration, consider a control strategy for solving a jigsaw puzzle: build the borders first, and then work in towards the center. This strategy exploits the local constraints provided by the straight edges of the border pieces to reduce the number of pieces that could be fit. Border pieces are less unconstrained and should be placed first; then, any piece that looks as if it might extend the frame should be placed next. A control strategy that exploits domain constraints this way can sometimes minimize uncertainty or its effects. AI systems use knowledge about uncertainty in their control strategies in a variety of ways. By recognizing those points where uncertainty is introduced, a control strategy can provide a mechanism to retract errorful conclusions or mark problematic issues for careful analysis. A control strategy can also concentrate on hypotheses (partially supported belief) with especially high or low certainty, or modify action on the basis of characteristics of uncertain evidence.

Dependency directed backtracking (Stallman and Sussman, 1977; Doyle 1979; London, 1978) is a method for efficiently recovering from errors in choices made with uncertainty. The behavior of a system can be seen as a tree, with each node representing a choice made under uncertainty. The power of backtracking is that the reasoning process assumes all nodes (choices) along a single branch of the tree are certain. When

[1]Something is typically considered known if it is immediately available in a database or if it can be found by some limited inference. But in some logic-based paradigms, something is known if it can be proven – deduced from the current set of assertions (Artificial Intelligence, 1980). See (Levesque, 1984) for a discussion of the difference.

A general assumption relating knowledge of a proposition to its truth is that X is true if and only if X is known. The contrapositive of implication in one direction (known(x) -¿ true(x)) is the closed world assumption as commonly understood. The positive implication in one direction (true(x) -¿ known(x) is the assumption made by (Collins, et. al., 1975) in plausible reasoning. The positive implication in the other direction (known(x) -¿ true(x)) is the assumption made by reasoning processes that ignore the effects of uncertainty in their beliefs, as in parallel certainty inference discussed in a later section.

a choice is found to be wrong, the reasoning process reconsiders and makes an alternate choice at that point. An efficient method for redoing the choice leaves the bulk of the belief set unchanged. A related approach, which records the *reasons* for the uncertainty at each choice point, is discussed below.

Least-commitment planning (Sacerdoti, 1977; Stefik, 1980) is a strategy to manage the uncertainty introduced by not knowing the effects of actions (i.e., incompleteness of the domain model). The construction of plan steps introduces uncertainty because possible interactions with other plan steps are not known in advance. By delaying the commitment to these plan steps until more interactions are known, the uncertainty in the effects on other parts of the plan is reduced.

Opportunistic control, as in the HEARSAY-II speech understanding system, (Erman, et.al., 1980) directs the system to focus its attention to those hypotheses that are supported with the greatest certainty, that is, to follow the most promising leads. These *islands of certainty* are sources of local constraints that make it easier to propose and support new hypotheses. In the speech understanding domain, the effects of uncertainty were minimized by this opportunistic strategy, which relied on the redundancy of information in the speech signal. One can imagine cases in which an opportunistic strategy is not as well-suited to the characteristic uncertainty of a domain. The point is that for the control approach to work, the control strategy must be matched to the kinds of uncertainty that arise in a domain.

Heuristic search can also benefit from the consideration of uncertainty. The term "heuristic knowledge" implies that the knowledge is imperfect (uncertain) in some way. Understanding the limitations of heuristic knowledge can be a source of power in using it. For instance, many computer chess programs incorporate a *static evaluator*, a heuristic that estimates the worth of a board position. By searching a few moves ahead and applying the static evaluator at the terminal nodes of the search tree to compare the relative worth of each move, a chess program can choose a reasonable move. A difficulty called the *horizon effect* (Berliner, 1974) occurs when beneficial positions are missed because the static evaluator is applied at a uniform depth. Important positions are missed if they are just over the horizon of the evaluator's view. A control strategy can improve performance if it extends the search at points where the horizon effect is most likely.

In summary, the control approach to uncertainty recognizes *where* uncertainty arises and incorporates a control strategy to provide flexibility at those points.

4.3 Parallel Certainty Inference

The parallel certainty approach divides the reasoning process into two semi-independent processes. One proceeds as if there were no uncertainty in its conclusions. The other decides on the certainty of the conclusions derived by the first. This is convenient because it allows the first process to concentrate on the domain problem without considering

difficulties introduced by uncertainty. The first process decides *what* to believe, and the second, *how much* or *why* to believe. Three broad categories of parallel certainty will be discussed: degrees of belief, reason maintenance, and the theory of endorsements.

Degrees of belief

The most popular parallel certainty methods represent uncertainty as a degree of belief,[2] an expression of *how much* something is to be believed. The canonical example is the *certainty factor* representation of MYCIN (Shortliffe and Buchanan, 1975) and PROSPECTOR (Duda, Hart, and Nilsson, 1976). A proposition is associated with a number between 0 and 1 that represents how much the system believes it. Inference rules are applied without regard to the certainty of their premises[3] (or conclusions, if the inference is backward-chaining), while degrees of belief are propagated from premises to conclusions via a rule of combination.

At least two sources of uncertainty are represented by certainty factors: certainty in inference rules and certainty in beliefs. A certainty factor for a rule represents the expert's confidence in it, but it is not always clear what confidence means. For example, a rule that states that obesity implies illness might have a certainty factor of 0.8 associated with it. This number might represent the proposition that 80that the probability is .80 that a sick person is obese, or that the general rule that obesity causes sickness is more applicable than a rule with a certainty factor of 0.6. Whatever its meaning, the *effect* of the certainty factor on a rule is to weight the belief in its conclusions; the higher the rule's cf, the higher the belief in conclusions from that rule (all things being equal). Certainty in beliefs is also represented by numbers. Again, it is difficult to be clear about what the certainty factor of a belief *means*, other than to say that higher numbers mean stronger belief.

Belief is propagated across inferences. The propagation rules used by MYCIN and PROSPECTOR are variants of Bayes' rule, which provides a mathematical method for updating the probability of a hypothesis given an observation of evidence. Bayesian methods are based on the axioms of probability theory, and have been applied in several ways to combine the degrees of belief for multiple hypotheses given evidence from multiple distinct sources that might disagree. They are quite general.

A related method of representing and reasoning with degrees of belief is the Shafer-Dempster method (Dempster, 1968; Shafer, 1976; Lowrance and Garvey, 1982). In contrast to the Bayesian approach, belief is represented by an interval between 0 and 1, rather than as a single point. The Shafer-Dempster method has a number of advantages over a strictly Bayesian approach, mainly because it makes weaker assumptions.

[2]The term is due to Shafer (1976).

[3]Actually, MYCIN did not fire rules whose conditions were believed with less than 0.2 cf, so it is not strictly a parallel inference method, since domain inferences are not kept entirely separate from inferences about uncertainty.

(Bayesians require all hypothesis to be mutually exclusive, exhaustive, singletons). The two-number representation allows for ignorance (the inescapable result of incomplete knowledge), as well as degree of belief, whereas in Bayesian models ignorance is commonly *mis*represented as belief in the negation. The Shafer-Dempster representation can capture belief in sets of hypotheses, which is particularly useful when uncertainty about the relevance of evidence prevents the assignment of belief to individual (singleton) hypotheses.

Many objections can be raised to representing uncertainty with degrees of belief. First, the semantics of the numbers are not well defined. Some authors interpret the numbers as subjective probabilities, others as frequencies, and others entirely ignore the issue of what the numbers mean. An emphasis of recent research (Rich, 1983; Kim and Pearl, 1983; Wesley, 1983; Ginsberg, 1984; Strat, 1984) has been to make numerical degrees of belief represent an increasing variety of *kinds* of uncertainty, so the interpretation of the numbers is a bewildering task. We believe that numbers are not an adequate representation for everything one wishes to say about the causes and consequences of uncertainty. A second problem, which is a consequence of the representational inadequacy of numbers, is that the numbers are *used* to represent combinations of factors; for example, certainty factors in rule-based systems frequently account for *salience* and *utility* as well as degree of belief. A third and related problem is that if the components of a degree of belief are unknown, or if their relative contributions are unknown, then it is impossible to predict whether transformations of degrees of belief – such as combining functions – have any effects on the meanings of the numbers, since the meanings were obscure to begin with. A rule may be given a high certainty factor because it is important, or useful, even if it is not very accurate. What interpretation does one give a number produced by combining two such certainty factors? A fourth problem, again closely related, is succinctly put in the question: "where do the numbers come from?". Salmon (1967) calls this the criterion of *ascerntainability*. How do we hope to effectively capture the knowledge of a human expert with numbers when the expert doesn't reason that way?

Reason Maintenance

Reason Maintenance (Doyle, 1980, 1983a). was developed specifically to deal with uncertainty caused by incomplete knowledge. Often, the truth of a proposition cannot be determined, but one can proceed as if it were known. Reason maintenance, and the theory of "reasoned assumption" most recently developed by Doyle (1983b), calls for jumping to conclusions in the case where the truth of a proposition is not known but can be assumed. In making assumptions of various forms, the system *consciously* introduces uncertainty; it records the *reasons* for the assumption, and thus represents sources of uncertainty associated with it. In terms of the parallel certainty inference model, the first inference process proceeds as if it has confidence in assumed propositions, and the

second provides a mechanism to carefully retract assumptions if they are found to be wrong. Thus, reasons for belief support sophisticated reasoning *about* uncertainty.

The Theory of Endorsements

The parallel certainty inference approach divides reasoning under uncertainty into two "streams"; one is a stream of logical inferences, typically the inferences that are needed to solve a problem. The other is a stream of inferences about the certainty of conclusions produced in the first stream. We have considered numerical inferences, based on Bayes' and Dempster's rules, and also reason maintenance – the recording and maintenance of dependencies among conclusions. A third approach is to record arbitrarily complex messages (which we call *endorsements*) in the second stream. These messages record causes and remedies of uncertainty; for example, we might note that evidence is produced by an occasionally faulty sensor, or that a newspaper reporter considers a wide range of sources before filing a report, or that the margin of error on a poll is 5%, or that a recommendation comes from someone who doesn't know his subject, and so on. The fundamental assumption of the *theory of endorsements* (Cohen, 1983) is that subjective degrees of belief, usually represented as numbers, are *composites* of reasons to believe and disbelieve. We suggest that, for many tasks, one needs to know more than simply the *extent* of one's belief; one also needs to know the *causes* of belief. The theory of endorsements is concerned with how to represent and reason with this knowledge.

We misrepresent the theory of endorsements somewhat by grouping it with parallel certainty inference approaches. One advantage of knowing why a proposition is uncertain is the ability to "take evasive action" to eliminate the cause, or the effects, of uncertainty. For example, if one knows that the cause of uncertainty is the absence of attainable knowledge, then one might eliminate the uncertainty by simply asking for the missing information, or by searching for it. On the other hand, if the missing knowledge isn't obtainable, then one cannot eliminate the cause of uncertainty but one may minimize its effects. For example, hedging minimizes uncertainty arising from unattainable knowledge. The key to such evasive action is knowledge about uncertainty. The search for missing evidence, for example, depends on knowledge about its source. If the source produces evidence intermittently, like a volcanic fault, then one must sit around and wait. We adopt one strategy for a feedback-directed search for evidence (e.g., we have found the right book, then the right section, and finally the desired sentence), and another for evidence that just "pops up" without warning (e.g., waiting for a bus that may or may not be running). Thus, the key to making a system responsive to its uncertainty is knowledge about the causes of uncertainty; or, conversely, parallel certainty inference approaches aren't responsive to uncertainty because they know nothing about it except its extent.

5 SOLOMON – An Implementation of the Theory of Endorsements

The theory of endorsements was initially developed in the context of rule-based systems, and was tested with expert heuristics from the domain of portfolio management (gleaned from a program called FOLIO; see Cohen and Lieberman, 1983). Our implementation of the theory of endorsements, a program called SOLOMON, reasoned about the uncertainty associated with these heuristics and their use. All such reasoning was mediated by structures called endorsements that represented reasons to believe and disbelieve their associated propositions. Endorsements are frame-like knowledge structures representing reasons to believe (**positive** endorsements) and disbelieve (*negative* endorsements). They are associated with propositions and inference rules at various times during reasoning. Five classes of endorsements appeared important for reasoning about uncertainty in rule-based systems:

Rule endorsements. Reasons to believe and disbelieve inference rules (e.g., a clause in a premise may be endorsed as **maybe-too-restrictive**, that is, the premise might occasionally fail due to this clause when the conclusion is in fact valid.)

Data endorsements. Reasons to believe and disbelieve raw data (e.g., a statement about one's own tolerance of risk is often **conservative**).

Task endorsements. Arguments about the evidence that executing tasks are likely to produce, used to schedule the tasks (e.g., a task is worth doing because it may produce a **corroborating** conclusion.)

Conclusion endorsements. Reasons to believe and disbelieve conclusions. These are combinations of a priori rule endorsements and detected relationships – such as corroboration – between conclusions (e.g., a **conservative** conclusion about one's risk tolerance is **corroborated** by other evidence.)

Resolution endorsements. Records of the application of methods to resolve uncertainty (e.g., no rules conclude a desired goal, but after eliminating a **maybe-too-restrictive** clause from a rule, we achieved the desired conclusion.)

The style of reasoning mediated by these endorsements is, by design, similar to the goal-directed reasoning of many expert systems: SOLOMON starts by trying to conclude a goal, usually the value of a parameter, such as risk-tolerance in the domain of investments. It then backchains through its rulebase, directed by this goal and its subgoals. As it proceeds, SOLOMON develops bodies of endorsements – reasons to believe and disbelive its conclusions. These provide justifications for the conclusions, and also play a role in the control of SOLOMON's reasoning.

It is important that endorsements should affect control of processing in SOLOMON, because the theory of endorsements is oriented towards the effects of uncertainty on behavior. In SOLOMON these effects were two: First, SOLOMON used endorsements to decide whether a proposition was **certain enough for the task at hand**. It would ask whether the endorsements of a subgoal conclusion were good enough to warrant using the conclusion to assert its parent goal. This is similar to setting a threshold on the numeric degree of belief that a conclusion must accrue in a backchaining system (e.g., MYCIN set a global threshold of 0.2.) However, the "threshold" is determined dynamically for each goal and applied to its subgoals' endorsements; and the threshold is not a quantity but a boolean combination of desirable and undesirable endorsements. Importantly, a proposition that is not certain enough for one task may serve for another; for example, the word of a used-car salesman might barely suffice if you want to know who won last night's football game, but is perilously untrustworthy where the salesman's self-interest is concerned.

The second effect of uncertainty on behavior is achieved, in SOLOMON, by **resolution tasks**. The principle of these tasks is that negative endorsements are viewed as problems to be solved. SOLOMON will attempt to improve the endorsement of an important proposition. It has available general and domain-specific rules for resolving uncertainty. For example, when it is unable to derive a desired conclusion from its available rules, it can make small modifications to the premises of the rules, such as dropping clauses. Clauses to drop are selected by their endorsements; SOLOMON will not drop clauses endorsed **as criterial**. Dropping clauses results in additional endorsements noting the uncertainty that it introduces (see Cohen, 1983, pp. 148-158, for a detailed example).

In addition to rules to decide when a proposition is certain enough for a task, and rules for resolving uncertainty, SOLOMON had a simple rule to combine endorsements and propagate them over inferences. This was that a conclusion inherits all endorsements of its premise, plus any that result from posting the conclusion (such as a contradiction between the conclusion and another). In fact, this rule was doubly flawed: First, reasons to believe or disbelieve a premise are **not** always endorsements of the conclusion; and, second, the rule led to large bodies of endorsements after only a few inferences. The remainder of this report reveals recent work on the problem of combining endorsements.

6 Combining Endorsements

Combining evidence is something that numerical approaches to uncertainty do very well, because they represent uncertainty as a quantity increased or diminished by evidence. We do not represent uncertainty as a quantity: We represent it in terms of knowledge about evidence, and we do not summarize this knowledge in a degree of

belief. Thus, it is not as easy to combine evidence in the theory of endorsements as it is in quantitative theories. If there is evidence from more than one source for a proposition, we must "calculate" a body of endorsements for the proposition by combining the endorsements of each piece of evidence. Simple syntactic union of the endorsements leads to the problems mentioned above: Large bodies of endorsements result, and not all endorsements remain relevant for all uses of their associated propositions. We are exploring **semantic combining rules** for endorsements – so called because the combination of endorsements is mediated by rules that reflect what the endorsements mean.

A related problem is **ranking** endorsements. Again, quantitative approaches can rank the credibility of hypotheses easily, and again, it is more difficult with endorsements. However, endorsements can be ranked on an ordinal scale, if not an interval one, and so schemes for ranking endorsements can be designed. This is the subject of a research note in preparation. This is the subject of the next section.

7 HMMM - An Endorsement-Based Plan Recognition Program

HMMM is a plan recognition program that infers which of several known plans a user intends by combining the evidence provided by successive user actions. Plan recognition is uncertain for two reasons: the user might make a mistake, in which case extrapolating from the action might suggest the wrong plan; or a user action may be ambiguous; i.e., the action might be consistent with several known plans.[4] If all the user's actions belong to only one known plan, the interpretation process is straightforward; but when an action can be interpreted as a mistake, or as belonging to more than one plan, HMMM is uncertain of the user's intentions, and so generates endorsements for the competing interpretations. HMMM is a simplified version of POISE (Carver, Lesser, and McCue, 1984), an office automation system with an intelligent user interface, which discerns a user's plan and offers assistance by automating some plan steps.

Individual plan steps are interpreted in the context of developing plans. The program uses its knowledge of the user's previous actions to restrict the interpretations of the current action. For example, assume the program knows the following plans:

Plan	Steps
plan1	a b d
plan2	b d e
plan3	a c d

[4]Other sources of uncertainty in plan recognition include an incomplete library of known plans and inaccuracies in the plan library. We limited our exploration to unintended and ambiguous actions.

Given that the user takes the actions a followed by b, we can construct three interpretations for each action:

(start plan1 a)	(continue plan1 b)
(start plan3 a)	(start plan2 b)
(mistake a)	(mistake b)

However, the interpretation of b as continuing plan1 would not be valid unless the first step of plan1, a, had already been taken. We account for these syntactic restrictions with data structures called *step linkages*. Each step linkage represents an interpretation of all the plan steps taken so far. Step linkages for the "current" step are constructed from the existing step linkages, which link all previous steps. For an interpretation that continues an already-opened plan (as b above continues plan1), each step linkage that mentions the preceding step is extended to include the new step. For an interpretation of a plan step as starting a new plan (as b above is interpreted as starting plan2), *all* step linkages are extended to include this interpretation.

Each step linkage carries endorsements. These are reasons to believe and disbelieve the interpretations of plan steps represented by the step linkages. For example, a reason to believe that b continues plan1, above, is that "continuity is desirable." Recall our contention that these reasons have no implicit meaning, no matter how evocative are the strings we use. The following example shows how meaning is ascribed to endorsements and how endorsements facilitate reasoning about uncertain interpretations.

7.1 An Example of Endorsement-Based Plan Recognition

Suppose we have a simple environment in which we know that the user intends exactly one of two known plans,

Plan	Steps
plan1	a b c
plan2	b d e

and the user types the input actions a followed by b followed by d. Briefly, we can imagine interpreting the first input as evidence for plan1, and the second as further evidence. The third input lends support for the plan2 interpretation of b, and casts doubt on the plan1 interpretation of a, and indirectly supports the possibility that a was a mistake. If a fourth input was c, we'd want the system to reaffirm its belief in plan1, whereas an input of e should have the opposite effect.

8 Applicability Conditions for Endorsements

HMMM uses endorsements to reason as just described. The actions a, b, d result in the following syntactic interpretations:

Step	Interpretation	Endorsements
1:a	(start plan1 a)	(a only grammatical possibility +) (a could be a mistake -)
2:b	(continue plan1 b)	(a b continuity is desirable +) (b other grammatical possibility -) (b could be a mistake -)
b	(start plan2 b)	(a b discontinuity is undesirable -) (b other grammatical possibility -) (b could be a mistake -)
3:d	(continue plan2 d)	(d only grammatical possibility +) (b d continuity is desirable +) (d could be a mistake -)

The endorsements are associated with the interpretations by rules specifying their applicability conditions: "other grammatical possibility" is applicable whenever a plan step figures in more than one possible plan; "could be a mistake" is always applicable; "continuity is desirable" is redundant with the interpretation of a plan step as continuing an open plan; and "discontinuity is undesirable" applies whenever a plan step is interpreted as disrupting an already open plan by starting a new one. Some endorsements are *positive*, meaning that they support the interpretation with which they are associated. Others are *negative* – reasons to disbelieve their associated interpretations.[5]

9 Combining Endorsements

The endorsements associated with an interpretation are brought along when that interpretation is appended to a step linkage, and they are combined with endorsements from the previous steps in the linkage to give the endorsements of the plan up to that point. For example, the input a is evidence for plan1, and b is *further* evidence for plan1. Note that b is a different kind of evidence from a, because it is ambiguous

[5] Applicability conditions for endorsements include rules to decide whether an endorsement is positive or negative. This is easy in HMMM, but we believe it to be difficult in general to decide whether evidence speaks for or against a hypothesis.

between plan1 and plan2. Applicability conditions for endorsements give us the mechanism to distinguish between the kinds of evidence - each kind carries characteristic endorsements - but they don't specify how to *combine* the endorsements of pieces of evidence, such as a and b, when they support the same hypothesis (in this case, plan1). To this end, we have implemented *semantic combining rules*, two of which follow.

SCR1: If (plan N: step i could be a mistake -) and
(plan N: steps i j continuity is desirable +)
Then erase (plan N: step i could be a mistake -)

SCR2: If (plan M: steps i j discontinuity is undesirable -) and
(plan M: steps j k continuity is desirable +) and
(plans N,M: step j other grammatical possibility -)
Then erase (plan M: steps i j discontinuity is undesirable -)

Both rules use the occurrence of two consecutive plan steps as a basis for removing negative endorsements that may have accrued to the first of the steps. The general idea is that consecutive steps in a single plan eliminate uncertainty about the interpretation of the first step. Given these rules, the *combined* endorsements for the plan1 interpretation of the inputs a, b and the plan2 interpretation of the inputs a, b, d are derived from the endorsed step linkages shown above:

plan1 interpretation of a, b:	**plan2 interpretation of a, b, d:**
(a only grammatical possibility +)	(b other grammatical possibility -)
(a b continuity is desirable +)	(d only grammatical possibility +)
(b could be a mistake -)	(b d continuity is desirable +)
(b other grammatical possibility -)	(d could be a mistake -)

Note that (a could be a mistake -) has been erased by application of SCR1 for the plan1 interpretation, and that (b could be a mistake -) and (a b discontinuity is undesirable -) have been erased by SCR1 and SCR2 respectively for the plan2 interpretation.

10 Strengthening Endorsements

The semantic combining rules discussed above are unintuitive because they eliminate endorsements entirely, rather than increasing or decreasing the weight of endorsements (e.g., a more intuitive version of SCR1 should reduce the concern that a plan step is a mistake, not drop it entirely). Currently, we use numerical weights to reflect the strengths of endorsements, and adjust the weights to reflect combinations of endorsements. Since we are concerned that these numbers should mean the same under

combination as combinations of endorsements, we have strictly limited ourselves to a single case of combination, namely *corroboration* of endorsements. We have identified three general situations where endorsements corroborate, that is, where two endorsements combine to create another "weightier" endorsement:

1. Corroboration of multiple instances of the same endorsement within a single plan step. For example, if an ambiguous plan step could continue one plan and start numerous others, then the weight of the "continuity is desirable" endorsement is greater than it would be if the step could continue a plan and start but a single plan.

2. Corroboration of instances of different endorsements of the same sign (both positive or negative) within the same plan step, resulting in a kind of synergetic increase in the belief in an interpretation. For example, the two negative endorsements "discontinuity is undesirable" and "other grammatical possibility" have a combined weight which is greater than the sum of their individual weights.

3. Corroboration of multiple instances of the same endorsement over consecutive plan steps. We believe in a plan more strongly if it is successively reinforced by the same positive endorsements. For example, we increase the weight of endorsements associated with a plan if the "continuity is desirable" endorsement appears in several consecutive steps.

11 Ranking Endorsements

We have said that the three components of semantics for endorsements are applicability conditions, combining rules, and ranking rules. We have explored two methods for ranking combinations of endorsements: one used the numerical weights of endorsements as described above, the other was a classification scheme to separate likely and unlikely alternatives.

We wanted combinations of endorsements to dictate at least a partial ordering on alternatives facing any decision-making program. We accomplished this in HMMM with a scheme for classifying step linkages as *likely*, *unlikely*, or *neutral*,[6] contingent on the presence of particular endorsements or combined endorsements. For example, a sufficient condition for being considered "likely" might be corroboration of two different, positive endorsements, and the condition for "unlikely" might be any negative endorsement. Interpretations can be ranked by assigning them to one of these implicitly-ordered classes, based on their endorsements. We think this kind of classification scheme can serve as a general model for ranking endorsements, since the criteria

[6]These terms are the names of classes; membership in any class is determined by endorsements. We imply no probabilistic interpretation of these terms.

for membership in classes are flexible (and may be set dynamically); and since the number of classes is also flexible, ensuring adequate discrimination of alternatives. (The classification scheme was originally devised for a planning program which predicts a planner's next move to be from the class of "likely" moves.)

12 Discussion

The HMMM program raises many questions about endorsement-based reasoning. Two we did not address in the body of this report concern the subjectivity and cost of endorsement-based reasoning.

Subjectivity of endorsements. Endorsement-based reasoning is not normative or prescriptive: there's no "correct" set of endorsements for a domain, no correct method for combining the endorsements of successive pieces of evidence. The endorsements discussed in this report seem appropriate to the domain of plan recognition. We believe that ambiguity of plan steps reduces certainty in all interpretations of those steps, just as certainty is increased when two or more consecutive steps are interpreted as belonging to the same plan. Other people might design a different set based on their perceptions of the domain. The point is that this report provides a framework for endorsement-based reasoning, but it is not prescriptive.

How much is required? The simple plan recognition example required few endorsements and only two semantic combining rules. We need more of each to handle other kinds of uncertainty and other relationships between endorsements. The number of endorsements and combining rules required for a domain depends on what you intend to do with them. If you wish to represent the *major* sources of uncertainty in a domain (e.g., the possibility of mistakes, ambiguity, disruption of an established scheme, etc.), then we believe the number of combining rules will be small. This is the approach we took for plan recognition. We expect that endorsements can constitute a small investment for system-builders with a large payoff in terms of explanatory power and facilitation of knowledge engineering (since the expert can give reasons for uncertainty instead of numbers).

To effectively reason *under* uncertainty, in the long run, intelligent systems must reason *about* uncertainty. This means specifying representations, thinking carefully about what they mean, developing operations for *combining* and *propagating* them, and considering what properties of uncertainty the operations preserve. Early work in reasoning with uncertainty concentrated on whether there was uncertainty and how much. This is adequate for some purposes, but the intelligent reasoning systems of the future will need richer representations for a more sophisticated approach to uncertainty. Some of the purposes to which sophisticated reasoning about uncertainty must be applied are explanation, evaluation, and control.

Explanation. We want to know "why" an agent believes something, not just

"how much" it is believed. Early inference systems such as TEIRESIAS, (Davis, 1976) explained their behavior by displaying the chain of rules leading to a conclusion; they didn't explain why those particular rules fired. In particular, they failed to explain the basis for partial support (i.e., certainty factors). It is not clear how a degree of belief summarizes the reasoning under uncertainty that produced it, and yet, it is precisely in conditions of uncertainty that good explanations are most beneficial.

Evaluation. AI systems cannot be evaluated as black boxes. Proper validation requires a consideration of the structure and content of internal belief. For example, Lenat's (1976) AM program discovers fundamental concepts in mathematics. That's the black box view. Only after several years of experiment did anyone (including Lenat) really understand why and how AM worked. (Lenat and Brown, 1983) That analysis, which demystified the original program and provided valuable insights into the nature of learning, was based on experiments with the structure and content of AM's representations. Similarly, we cannot hope to understand how our systems reason under uncertainty unless we "open up" the black box representations of uncertainty. As with AM, we can say that our systems "work." But they do not currently give us any insight into the sources and consequences of uncertainty.

Control. Most expert systems use relatively simple control strategies. Processing is data-driven or goal-driven, or the two may be mixed in an opportunistic manner. Focus of attention in opportunistic systems is managed by numerically weighing, in empirically derived equations, alternative actions (e.g., Erman and Lesser, 1980). Unfortunately these numeric assessments hide the reasons for performing one action over another. We propose that flexible control strategies for reasoning in uncertain domains must be sensitive to the causes and consequences of uncertainty. Only if these are represented explicitly, can a system tailor its actions to minimize uncertainty or its consequences.

In conclusion, sophisticated reasoning about uncertainty will require adequate representations of knowledge about the causes and consequences of uncertainty, and adequate mechanisms for weighing, combining, and selecting actions, based on these representations.

Chapter 2

Semantics of Endorsements and the GRANT Project

The work on endorsements was eventually impeded by a difficult question: Where do endorsements come from, and what do they mean? The mnemonic value of endorsements like **may be a mistake** disguises the fact that endorsements are arbitrary symbols, whose meaning comes from the rules by which they are combined with other endorsements. We were concerned that, for complex domains, dozens of endorsements and combining schemes would have to be acquired. Although we had no objection in principle to acquiring this knowledge from an expert (much as other domain knowledge is acquired), we wondered whether the endorsements and combining schemas of a domain could be derived from other knowledge about the domain, such as inference rules. If so, we would worry less about whether we had the "right" endorsements and combining schemas.

We focused on the uncertainty inherent in a single problem-solving task, namely classification, to pinpoint the sources of uncertainty (and thus endorsements) of all classification tasks. Classification is the problem solved by many or most expert systems (Clancey, 1984).

Uncertainty in classification problem solving has two major sources. The first is that data may be inaccurate or incomplete, and the second is partial matching. This article is not concerned with the quality of data; we focus instead on uncertainty inherent in the design and behavior of classification systems. The partial matching problem has two forms, easily illustrated by the following common, empirical association: *A person with a queasy stomach, fatigue, aching limbs, and a fever has flu in its early stages.* Now consider a person with a marginal fever, complaining of poor appetite, headache, and a persistent twitch in his left eye. This case seems to exhibit manifestations not stated in the rule for flu and fails to display manifestations that are so stated. We are uncertain whether the person has flu for two distinct reasons: we cannot be certain that the actual symptoms fail to match the stated ones (Does "marginal fever" count

as a fever? Does "headache" count as aching limbs?); and we cannot be certain that the rule for flu includes all and only the relevant manifestations of flu.

We suggest that the interpretation of probability in classification systems should be in terms of similarity, not in terms of games of chance. This interpretation has precedent in some frame-based expert systems (*e.g.*, PIP and INTERNIST) and in psychological literature, where it is called the *representativeness* heuristic:

> Many of the probabilistic questions with which people are concerned belong to one of the following types: What is the probability that object A belongs to class B? What is the probability that event A originates from process B? What is the probability that process B will generate event A? In answering these questions, people typically rely on the representativeness heuristic, in which probabilities are evaluated by the degree to which A is representative of B, that is, by the degree to which A resembles B. (Tversky and Kahneman, 1982, p. 4)

Assessments of subjective probability in classification situations are insensitive to factors that affect probability (such as prior probability distributions) and sensitive to the resemblance between data and their classification. For example, Kahneman and Tversky asked subjects to classify individuals as librarians or truck drivers on the basis of personality sketches. They found that the classification was insensitive to the prior distribution of librarians and truck drivers in the population. An individual described as "neat, methodical, and shy" was classified as a librarian even if the prior probability of being a librarian was low. Remarkably, subjects ignored prior probability even when the personality sketches were completely uninformative, assessing a probability of 0.5 for each alternative instead. Translating these results to the expert systems literature, we would expect degrees of belief in heuristic associations between data and solutions–often represented as conditional probabilities–to be interpreted not in terms of relative frequency, but in terms of the degree to which data are representative of a solution. We might hope that experts would use probabilistic information more efficiently than novices, but evidence suggests that experts are as prone to judgment by representativeness as the rest of us (Kahneman and Tversky, 1982, p. 35).

Intuitively, the degree to which evidence is representative of a conclusion determines the credibility of the conclusion given the evidence. But if representativeness is to be useful as an interpretation of uncertainty in AI programs, we need a way to measure it.

The concept of representativeness is described only informally in the psychology literature. An obvious implementation of representativeness, discussed in Section 2.2, calculates the degree to which an instance is representative of a class by a weighted sum of their common properties. For example, we say a person is likely to be suffering flu if he or she has relatively many flu symptoms (properties) and relatively few non-

flu symptoms [1]. This intuitive approach — counting common properties — fails if an instance shares semantically-related, but nonidentical properties with a prototype. Imagine that the prototype for flu includes the property "nausea," but the patient reports "loss of appetite"; or the prototype may include "aching limbs," and the patient reports "pain across the neck and shoulders." In these cases, we are obliged to look at the degree of semantic match between properties before we can calculate the total degree of match between two concepts.

In fact, we believe there are six sources of uncertainty in classification by partial matching: Solutions are uncertain when data may be inaccurate; when the prototypes (e.g., rules or frames) may be incorrect; when one cannot find data to match an aspect of a prototype; when a prototype fails to account for some data; and when the procedures that match data with prototypes make errors of ommission or commission, that is, when the procedures fail to match relevant data to a prototype, or when they match irrelevant data to a prototype. These six cases are shown in Figure 2.1, and illustrated in the context of an example· A researcher is applying for funding for work on the effects of dietary sodium on heart disease in tribal African populations. Two possible funding agencies describe their interests as

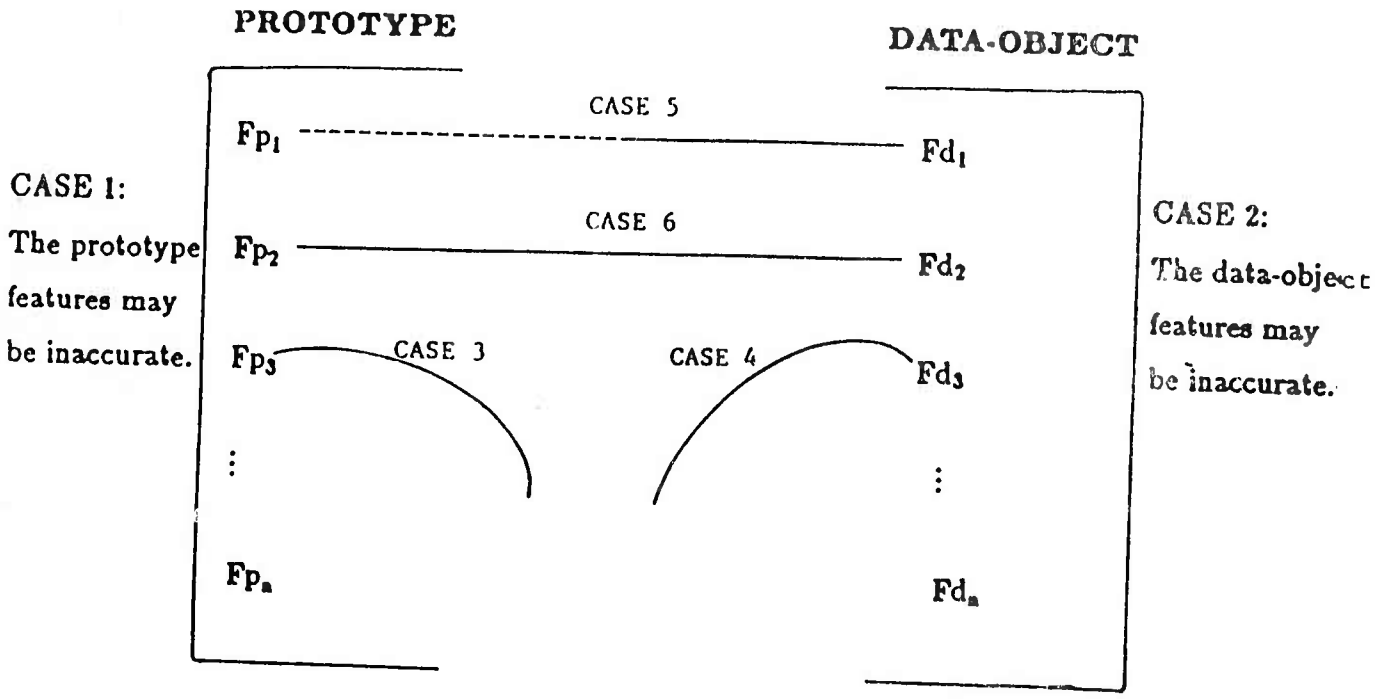
1. funding new investigators to research the effects of diet on health

2. the effects of dietary sodium on appetite

Consider the proposal to be data, and the agencies to be prototypes. Assume, for the moment, that a program matches the data with features of the prototypes as shown in Figure 2.2 and concludes that agency 1 is more likely than agency 2 to fund the proposal. Here, the word "likely" reflects the degree of match between each agency and the proposal: agency 1 is considered the better match. This conclusion is uncertain for the six reasons just articulated: 1) agencies 1 and 2 may not have described their interests correctly, 2) the researcher may have described her interests incorrectly, 3) agency 1 wants to fund new investigators, but no datum matches this feature, 4) neither agency accounts for the datum that the research is to be done in tribal african populations. These four problems are well-known (e.g., Hayes-Roth, 1978; Tversky, 1977), and many schemes have been proposed to deal with them. To understand the last two sources of uncertainty, note that we *assumed* the matches between data and features that are illustrated with dotted lines in Figure 2.2. One match – between dietary sodium in the proposal and agency 2, is exact. The others are *semantic* matches. They assume a semantic memory in which associative paths (represented by dotted lines) hold between diet and dietary sodium, health and heart-disease, and appetite and heart disease. A matcher unable to exploit these associative

[1] Clearly, the representativeness interpretation of likelihood is not probabilistic in the frequentist or Bayesian flu — only the number of shared and unshared symptoms — in assessing the liklihood of flu. See Tversky and Kahneman (1982) for other examples.

FIGURE 2.1



CASE 1: Fp_i may be inaccurate.

CASE 2: Fd_i may be inaccurate.

CASE 3: No match found between Fp_3 and Fd_i.

CASE 4: No match found between Fd_3 and Fp_i.

CASE 5: This match should have occurred but didn't.

CASE 6: This match should not have occurred but did.

paths (i.e., a syntactic matcher that requires equality of the objects to be matched) would fail to match diet with dietary sodium. This would be an example of case 5, above – a datum may match a feature in the sense that an associative path connects them, but the matcher doesn't report the match. However, in principle, one can find an associative path – often lengthy and indirect – between *any* datum and *any* feature of a prototype, if these objects are nodes in semantic memory. The presence of an associative path does not guarantee a "good" semantic match. Intuitively, the match between appetite and heart disease seems to be based on such a path, and is an example of case 6, above – a datum inappropriately matched with a feature. Allowing semantic matches, then, introduces uncertainty that data will be inappropriately matched with features (case 6); but any attempt to restrict semantic matching introduces uncertainty that an appropriate match has been disallowed (case 5).

In this report, we ignore cases 1 and 2 altogether, assuming accurate data and accurately-specified prototypes. We address cases 3 and 4 this way: a feature of a prototype lacks a match if no *credible associative path* can be found between a datum and the feature; and a datum is unaccounted-for if no credible path can be found between it and any feature of the prototype. Assuming criteria for what constitutes a credible path, the credibility of a match between several data and a prototype depends on how many features are unmatched with data (case 3) and how many data are unmatched with features (case 4). Cases 5 and 6, then, reflect uncertainty about what constitutes a credible associative path. Case 5 reflects concern that the criteria for a credible associative path are too stringent; case 6, that they are too lax. Said differently, cases 5 and 6 reflect concern that data should, or should not, be considered *evidence* for a prototype. These cases are the main concern of this report.

Our central claim is that the degree to which a datum provides *evidential support* for a prototype depends on the associative pathways between the datum and the features of the prototype in a semantic memory. Once we know whether data support individual features of prototypes, we can ask how many features are supported, and derive some measure of the overall fit between data and a prototype. This is illustrated in Figure 2.2: the proposal seems a good match to agency 1 because there seem to be semantic matches between two of the features of the agency and data from the proposal (although another datum–tribal african populations–is unaccounted for). Diet matches dietary sodium and health matches heart disease. Agency 2 seems poorly matched (even though it shares with the proposal an interest in dietary sodium) because no apparent match holds between appetite and heart disease. However, if we knew that appetite was strongly associated with heart disease (perhaps as a symptom), then the match between the proposal and agency 2 would seem stronger.

Given this claim, if we knew which pathways provide evidential support between any datum and prototype feature, then the uncertainty of cases 5 and 6 could be eliminated. Our approach to managing this uncertainty, discussed later, is to mark a small number of general pathways as particularly likely or unlikely to provide evidential support.

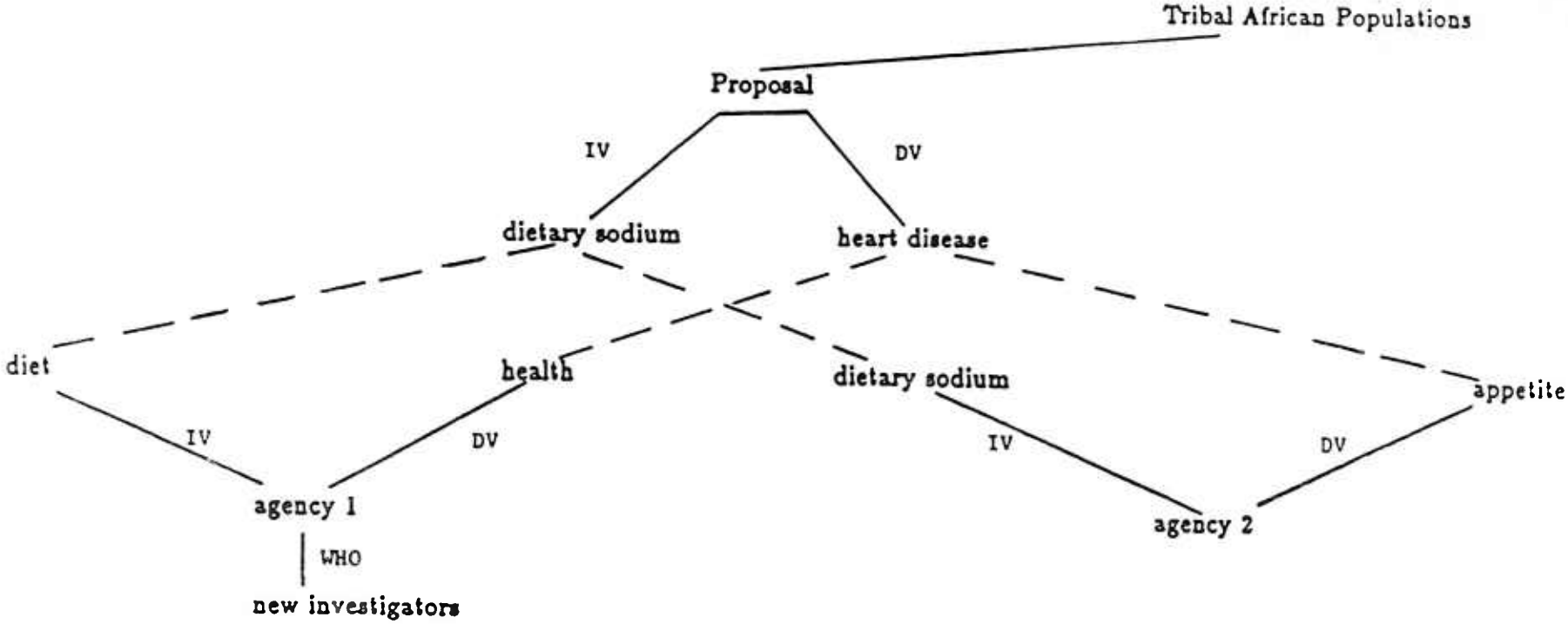

Tribal African Populations
Proposal
IV
DV
dietary sodium
heart disease
diet
health
dietary sodium
appetite
IV
DV
IV
DV
agency 1
agency 2
WHO
new investigators

FIGURE 2.2

In the following sections, we describe a particular matching task in overview, then formally specify the knowledge representation language used by an expert system that performs this task. We then discuss the performance of the expert system as a test of our claim that the degree of evidential support between data and prototypes depends on the nature of the associations between them.

1 GRANT

GRANT is a knowledge system that finds sources of funding for research proposals. The user builds a representation of a research proposal and instructs GRANT to search for funding agencies that are likely to provide support. GRANT first constructs, then ranks, a *candidate list* of agencies. An agency is added to the candidate list if a single topic in its statement of interests is a good semantic match to a topic in the research proposal. Semantic matches exist between topics that are the endpoints of particular *paths* through a semantic network. Agencies on the candidate list are ranked by the number of semantic matches between all the topics in the proposal and all the topics in each agency's statement of interests. The best-ranked agencies are thus those that support the largest number of topics that are semantically related to the proposal.

The key assumption of the system is that if no agency can be found to support research on a specific topic, then one might be found to support work on a semantically-related topic, and the likelihood of support depends on the relationship between the topics. Imagine a researcher is interested in dandelions, but GRANT cannot find any funding agencies in its memory that mention dandelions. GRANT may, however, find an agency to fund research on a *related* topic, say plants. The likelihood that the agency will fund work on dandelions depends, in part, on the nature of the relationship between dandelions and plants. Once GRANT has found an agency to fund a given topic or a related one, it then calculates how well all aspects of the agency description fit those of the research proposal. These two phases, finding an agency and computing overall match, are the main components of GRANT. Since the novel aspect of GRANT's architecture is how it finds agencies, that will be the focus of this report.

2 GRANT Architecture

GRANT's architecture includes a large semantic network of research topics, a set of funding agencies, a user interface for specifying proposals and presenting results, and a control structure for finding agencies given a proposal. These are illustrated in Figure 2.3. The semantic network is in effect an *index* to the agencies, since each agency is linked into the network at those nodes of the network that represent its research interests. Proposals, once elicited from researchers, are linked into the network in the same way.

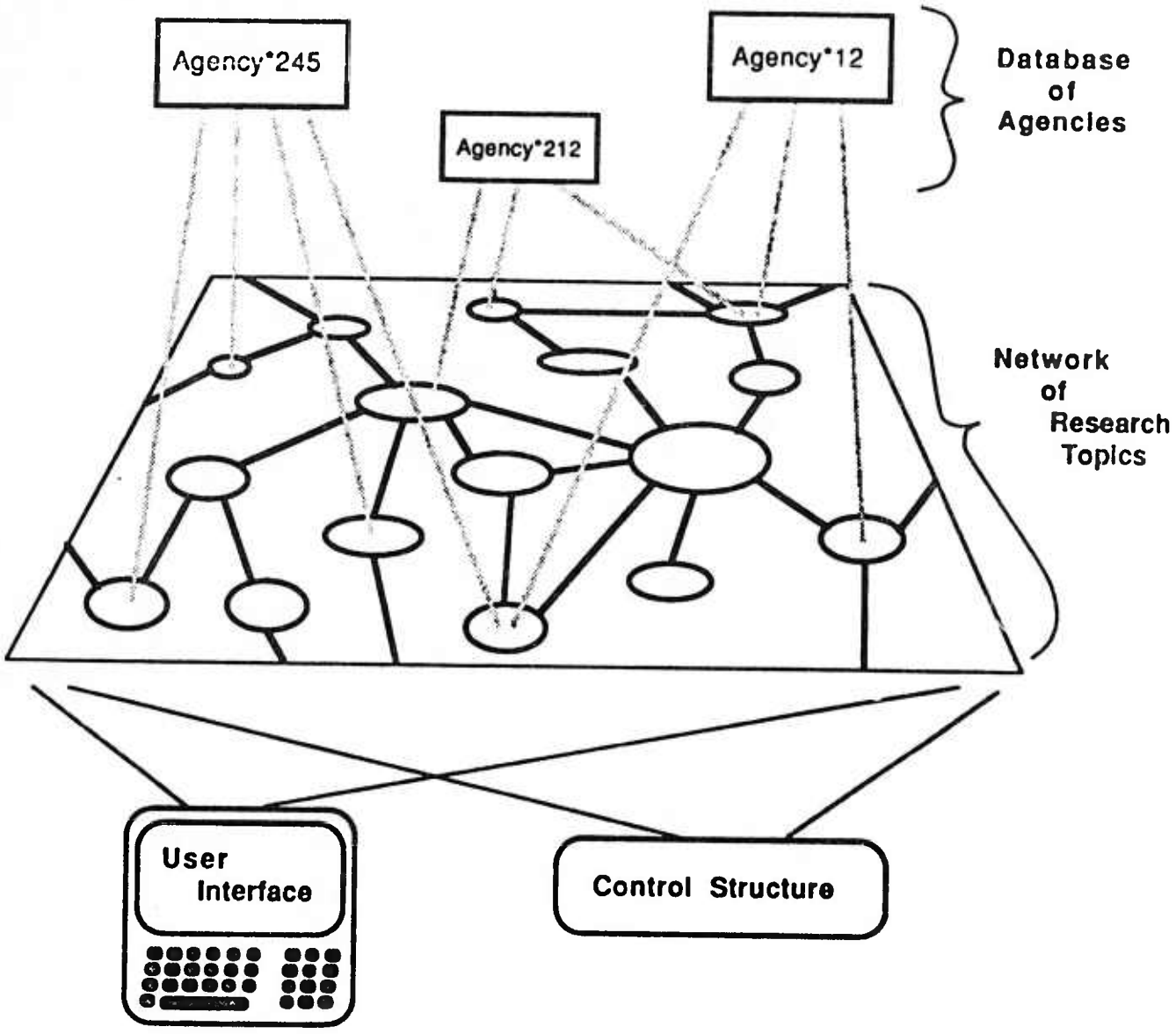


FIGURE 2.3

Overview of the GRANT System

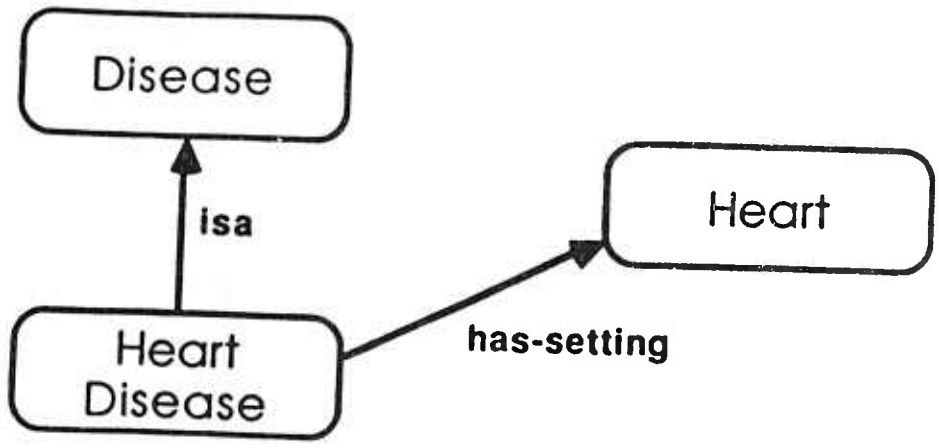


FIGURE 2.4

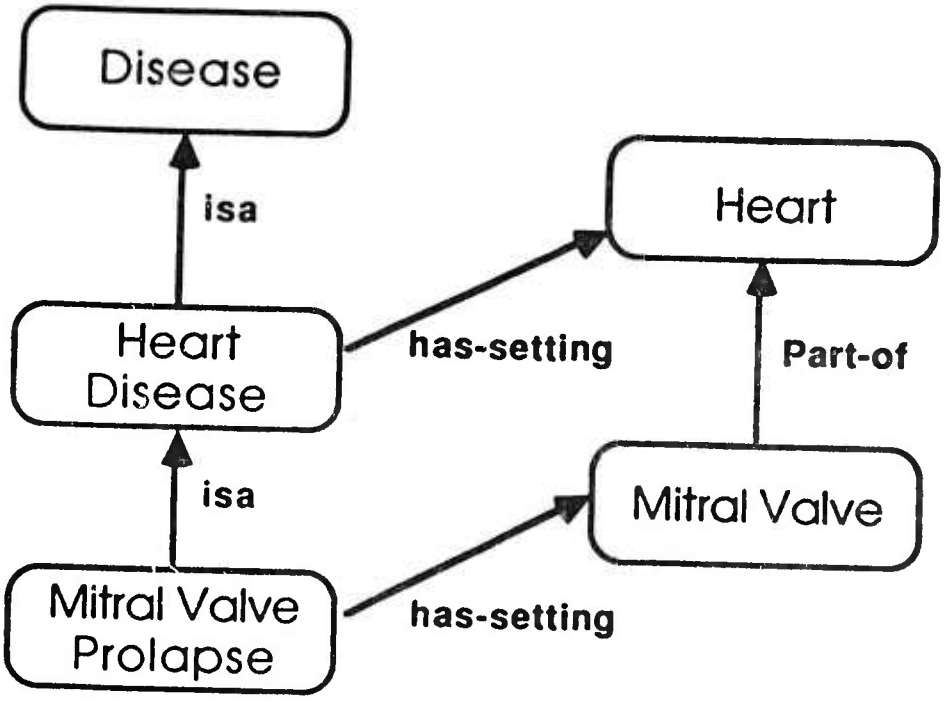


FIGURE 2.5

In overview, the system works by *spreading activation* from a proposal through the network until one or more agencies are activated. First, the research topics in a proposal are activated, followed by all topics that are directly related (i.e., one link away) in the network, followed by *their* related topics, and so on, as activation spreads across relations in the network like ripples in a pond. Ordinary spreading activation can quickly touch every topic in a network, which means that it can find pathways from any research proposal to any agency description. Since most agencies found this way would not fund a given proposal, GRANT uses a modified search algorithm, called *constrained spreading activation.* This algorithm is constrained by a set of rules to favor particular pathways through the network, and terminate search along other pathways. The rules lead GRANT to agencies that cannot be found by keyword search, and allow it to avoid the numerous, irrelevant agencies that are found by ordinary spreading activation.

2.1 GRANT's Knowledge Base

. GRANT's semantic network of research topics was constructed specifically to represent the interests of funding agencies. Currently, the network contains over 4500 nodes that represent the research interests of 700 funding agencies. Nodes are added to the network by linking them to other nodes with one or more of 48 distinct relations. For example, we can define a *heart disease* node by linking it to *heart* with the *has-setting* relation and to the *disease* node with the *isa* relation (see Figure 2.4). All relations are directional and have inverses (not shown in Figure 2.4); for example, the inverse of *has-setting* is *setting-of* and the inverse of *isa* is *has-instance.* GRANT adds inverse links between nodes automatically.

Sometimes the nodes that would define a new node do not exist in the network and must themselves be defined. For example, to add *mitral valve prolapse* to Figure 2.4 we need to say it is a *heart disease* but we also need to say its setting is the *mitral valve*, which is part of the *heart.* Figure 2.5 shows how adding *mitral valve prolapse* also involves adding *mitral valve.* Nodes are added only as needed to define research topics; GRANT's knowledge base is not an encyclopedia of science, medicine, and the arts, but is a highly cross-referenced index of research topics, represented from the perspective of funding agencies[2].

The relationships that define concepts are similarly tuned to GRANT's domain; for example, one field of research is a *subfield* of another, a phenomenon is an *effect* of a process, something is a *dependent variable* of a study, and so on.

All nodes in the network are represented as frames. Slots represent links or relations with other nodes. Some nodes represent funding agencies and the research topics they

[2]See Lenat, Prakash, and Shepard (1986), for a fascinating description of an encyclopedic knowledge base.

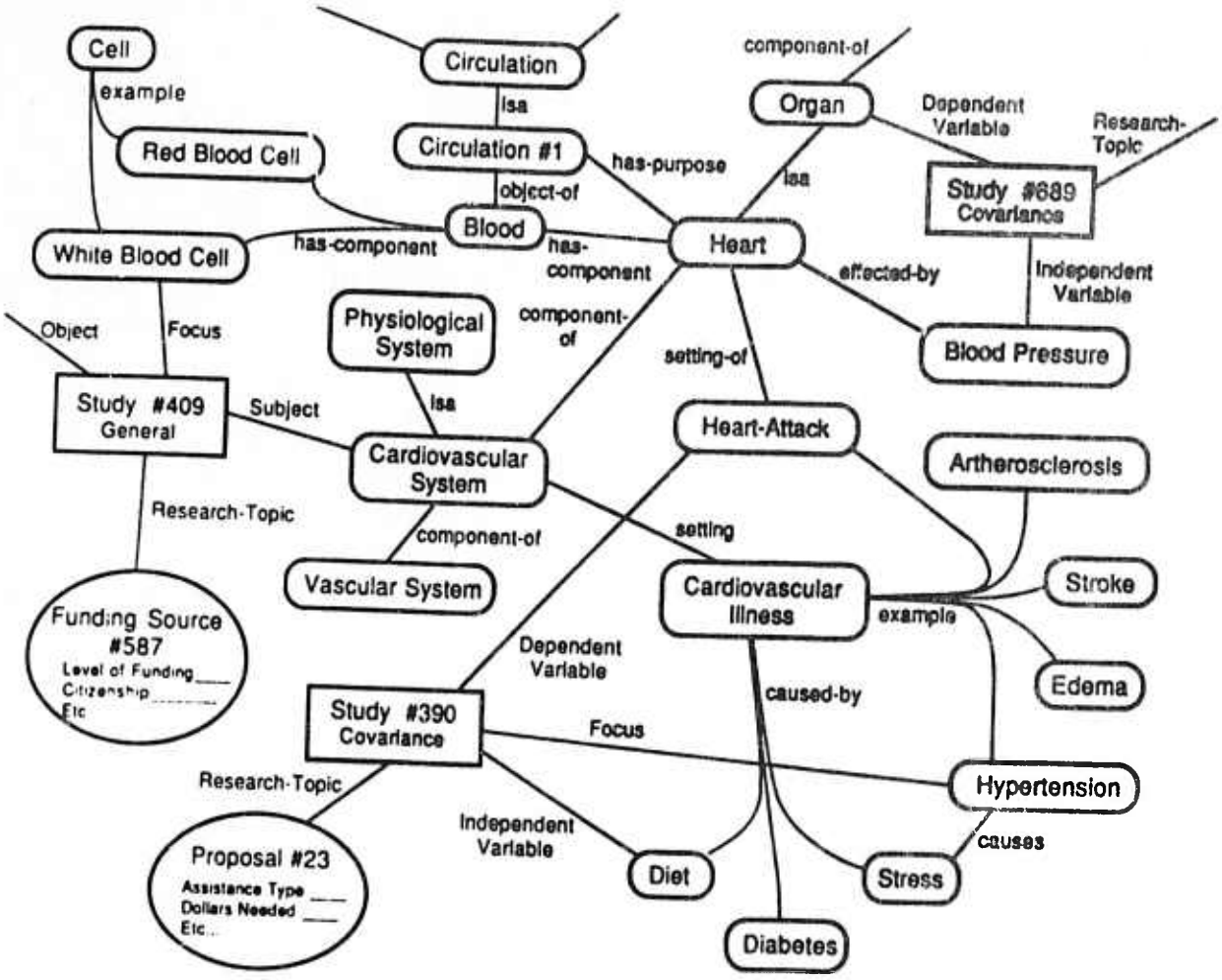


FIGURE 2.6: A portion of the GRANT knowledge base

support. Agencies have slots for level of funding, citizenship restrictions, and so on, as well as links to their research interests (Figure 2.6).

The frames that describe research interests, both for agencies and proposals, are created by classifying the goal(s) of research into one or more of ten classes:

Design	Educate	Improve	Intervene	Manage
Supply	Promote	Protect	Study	Train

Each class is represented by a case frame with a set of obligatory and optional slots. For example, a *study* frame represents exploration of some topic, and so has *subject* and *object* slots that represent the topic, and a *focus* slot that describes which aspect of the topic will be studied.

2.2 Constrained Spreading Activation

During a run of the GRANT system, activation spreads from the topics stated in a proposal, through the network, to agencies via their stated interests. Some constraint on the spreading activation is required, otherwise all agencies linked into the network would eventually be activated. Three kinds of constraints have been imposed. The

distance constraint says that activation should cease at a distance of 4 links (i.e., 5 nodes) from any research topic mentioned in the proposal. This is an extremely weak constraint. A second *fan-out* constraint says that activation should cease at nodes that have very high connectivity or fan-out. Examples of these nodes include *science*, *disease*, and *person*. Two research topics may be semantically related by both being sciences, but this does not guarantee that an agency will fund one if it will fund the other.

The third kind of constraint captures the idea that the likelihood of an agency funding a proposal depends on the nature of the relationships between the agency's interests and those of the researcher. Formally, GRANT is an inference system that applies repeatedly a single inference schema:

request-funds-for-topic(x) and **R**(x,y) → request-funds-for-topic(y) (1)

for "paths" **R**. (Note that **R** can be thought of as a single link, such as ISA, or more generally as a path of n links connecting n + 1 nodes, as described below.) If one would ask an agency to fund research on dandelions, request-funds-for-topic(dandelions), and dandelions are a kind of plant, then one stands a reasonable chance of obtaining funding from an agency that supports research on plants.

request-funds-for-topic(dandelions) and ISA(dandelions,plants) →

request-funds-for-topic(plants) (2)

If we replace the constants with variables, leaving just the relationship ISA, we get a rule of inference of the form described in (1) that we call a *path endorsement*:

request-funds-for-topic(x) and ISA(x,y) →

request-funds-for-topic(y) (3)

Associated with each path endorsement is a score denoting how likely it is that an agency would fund research on topic x if they would fund research on topic y. The rule above has a high score because funding agencies often support work on specializations of their stated interests; an agency may specify *plants* but support *dandelions*, may specify *transportation* but support *air travel*, may specify *heart disease* but support *mitral valve prolapse*. On the other hand, agencies typically state their interests at the most general level possible, so proposals that request funding for more general topics are likely to be denied. One cannot approach the National Heart, Lung, and Blood Institute with a proposal to study anatomy, since that agency is interested in much more specialized topics. This reasoning is represented by giving the following path endorsement a low score, and calling it a *negative* path endorsement.

request-funds-for-topic(x) and INSTANCE-OF(x,y) →

request-funds-for-topic(y) (4)

Negative path endorsements constrain spreading activation by disallowing particular transitions through the network. The example in (4) says that if we are searching for funding from the *heart-disease* node in Figure 2.5, we should not allow activation to spread to the *mitral valve prolapse* node over the *instance-of* relation because any agency associated with that node would be unlikely to fund the proposal.

The relationship **R** in (1) need not be a single link, but could be a chain of links. Referring again to Figure 2.5, one can imagine that a funding agency interested in the heart might support work on mitral valve prolapse; that is, spreading activation from mitral valve prolapse to its *setting*, the mitral valve, then to the heart, which *has-part* mitral valve, may find an agency that is likely to fund the original proposal. This is denoted by giving a high score to the positive path endorsement

request-funds-for-topic(x) and HAS-SETTING:PART-OF(x,y) →
request-funds-for-topic(y) (5)

Negative path endorsements like (4) constrain search by disallowing spreading activation. Since GRANT follows high-scoring endorsed paths before lower-scoring ones, positive endorsements like (5) order search. Path endorsements are *heuristic*: (3) and (5) could lead to agencies that will not fund the proposal, and (4) could lead to a willing one[3]. Currently, GRANT uses about 120 path endorsements to prune and order search paths. These were determined empirically during the early days of the GRANT project and have not been changed appreciably since. Given that 48 different links are used in GRANT's network, many more than 120 different pathways can be traversed. The set of path endorsements is not complete, except in the weak sense that unendorsed pathways are treated as if they are negatively endorsed – that is, they are pruned during search.

The matching of the previous section is accomplished in a **knowledge network**, formally described as a collection (O, L, P, D) where

O is a set of structured **objects**.

L is a set of binary relations between objects called **link-types**. Each link-type $l \in L$ links two objects.

P is a set of distinguished objects called **prototypes**.

[3]GRANT engages in *best-first search* (Nilsson, 1980) through a search space defined by its network. The heuristic evaluation function is not computed dynamically at each node by lookahead, but is rather a precompiled list of endorsed paths to search and prune.

D is a set of distinguished objects called **data-objects**.

A **link** is a triple $o_1 l o_2$, with $o_1, o_2 \in O, l \in L$.

A **feature** of object o, $f(o)$, is a link from o to some other object: $f(o) = o_1 l o_2$, where $o = o_1$ or $o = o_2$.

An object o is a "frame" uniquely defined by its features:

$$o = f_1, f_2, \cdots, f_n$$

A **path** between object o_{start} and o_{end} is a sequence of links connecting o_{start} to o_{end}:

$$\text{Path}(o_{start}, o_{end}) = \langle o_{start} l_1 o_1, o_1 l_2 o_2, \cdots, o_{k-1} l_k o_{end} \rangle$$

A **path endorsement** is a generalization of a set of paths:

$$l_1 l_2 = \langle D_i l_1 0_j l_2 P_k \rangle$$

$l_1 l_2$ is the path endorsement of a path between any Data-object D_i, linked by l_1 to any object O_j, linked by l_2 to any prototype P_k.

Path endorsements thus represent the associative pathways between data and prototype, without regard to the identity of data objects, prototypes, or objects intermediate on the pathways.

We claim that the degree to which a feature of data provides evidential support for a feature of a prototype depends only on the endorsement of the path that connects them.

GRANT performs a best-first search through its knowledge base, guided by path endorsements. Assume the program starts at a proposal and follows a link to an object: $\langle \text{Proposal}_{start} l_1 o_1 \rangle$. If a continuation of this path, $l_i o_i$, results in a path endorsement $l_1 l_i$ that GRANT recognizes as poor, then o_i is pruned from the list of nodes that GRANT tries to expand. If $l_1 l_j$ is a good path endorsement, then GRANT will give o_j priority to be expanded before any node o_k found by an unknown path $\langle \text{Proposal } l_1 o_1 l_k o_k \rangle$.

Constrained spreading activation finds a single semantic pathway between a proposal and each agency it reports as a potential funding source. But what if the proposal and agency share just a single interest – discovered by the search – but are otherwise completely different? For example, an agency may support research on reproduction in plants, while a proposal requests funding to study the economic impact of dandelions on landscaping. These seem to be a poor match, yet according to (2) above, the agency is likely to fund the proposal based on the semantic match between *dandelions* and *plants*. It appears that GRANT needs a way to calculate the full match between all aspects of a proposal and an agency, once it has found a partial match based on single pathway between them.

The result of best-first search is a candidate list of agencies. Each is known to have a single research interest that atomically or semantically matches one research interest

of the proposal. To the extent that the proposal and an agency share several common research interests, the agency is more likely to fund the proposal. Thus, GRANT ranks the candidate list of agencies by the degree of overlap between the research interests of the proposal and each agency. This is done by a partial matching function based on both atomic and semantic matching. Hayes-Roth (1978), Tversky (1977), and others measure the degree of overlap between sets in terms of set intersection and symmetric difference; for example, Tversky's *contrast model* (1977) calculates overlap this way:

$$S(a,b) = \theta f(A \cap B) - \alpha f(A - B) - \beta f(B - A).$$

The function f returns the cardinality of the set to which it is applied. If A and B are frames, then f(A $\cap$ B) is the number of slot-value pairs shared by A and B, and f(A $-$ B) is the number of slot-value pairs in A not shared by B. The parameters θ, α, and β are set empirically; in GRANT each is 1.0. If A and B are frames representing the research interests of a proposal and an agency, respectively, then $S(a,b)$ measures the number of research topics they have in common relative to those they do not share. Agencies for which $S(a,b)$ is higher are more likely to fund the proposal.

In GRANT, (A $\cap$ B) includes both atomic and semantic matches. If a path between A and B contains a single node (e.g., the first case in Figure 2.5), or if the path is an instance of a likely path endorsement (e.g., the second case in Figure 2.5), then f(A $\cap$ B) is incremented. Unlikely path endorsements, such as the third case in Figure 2.5, and unknown paths do not contribute to f(A $\cap$ B). The quantities f(A $-$ B) and f(B $-$ A) are increased when research topics in the proposal lack an atomic or semantic match to the agency, and vice versa.

In fact, we have not focused on full matching algorithms because GRANT currently performs adequately without one, and because its performance was not significantly improved when we added one to an earlier version of the system. Looking to the future, however, the analyses of partial matching presented in this report have convinced us that GRANT will eventually require full matching to achieve major reductions in its fallout rate.

3 Evaluation of GRANT

GRANT's evolution from a small, prototype system (Cohen, et al., 1985) to the present has given us the opportunity to compare performance as the system has been scaled up, and to consider the potentials and pitfalls of developing other GRANT-like systems. This section discusses a battery of tests on the current system.

The primary measures of GRANT's performance are *recall* and *fallout rate*. (A third statistic, *precision*, is 1.0 - fallout.) Recall is the percentage of all the agencies accepted by the expert that GRANT found, and fallout is the percentage of all the agencies found by GRANT that were judged good by GRANT but bad by the expert:

$$\text{fallout} = \frac{\text{num. of agencies judged good by GRANT, bad by expert}}{\text{num. of agencies judged good by GRANT}}$$

$$\text{recall rate} = \frac{\text{num. of agencies judged good by GRANT, good by expert}}{\text{num. of agencies judged good by expert}}$$

To calculate recall and fallout for a proposal, we need to generate a list of agencies from which the expert can select the ones that are likely to fund the proposal. One method would be to have the expert rank all 700 agencies in the network for each proposal, but this would be exhausting. Instead, GRANT is run in a minimally-constrained, spreading activation search that reports all agencies found within a given "distance" from each research topic in the proposal. This is called breadth-first (BF) search[4]. For each proposal, we first run a BF search then ask our expert to classify the agencies it finds as good or bad. Since the search is blind, many of the agencies are bad; that is, unlikely in the expert's judgment to fund the proposal. Then we run GRANT in an *endorsment constrained* mode called EC search, avoiding negatively-endorsed pathways and favoring positively-endorsed ones. It finds a subset of the agencies discovered by BF search. Ideally, it should find all and only the agencies ranked as good by the expert, but in practice it fails to find some of the good agencies (called *misses*) and finds some bad ones (called *false positives*). GRANT's miss rate tends to be very low, so we will be concerned primarily with the relationship between the fallout rate and recall rate.

The following tests were all performed on a set of 27 proposals, representing the interests of a diverse group of first-year faculty at the University of Massachusetts. The first test was designed to probe the utility of endorsement-constrained search. We compared EC and BF search with a third mode called *unconstrained keyword search* (UKW). It finds all agencies that share a common research interest with a proposal. It is implemented as a search for all agencies exactly 2 links distant from the proposal. For example, if a proposal and an agency share the common interest *dandelions*, then each will be linked to that node by, say, a SUBJECT link. The two-link

SUBJECT : *dandelions* : SUBJECT-OF

path connects the agency and the proposal via the common term dandelion; and, in general, any two-link path between an agency and a proposal indicates a shared term. UKW search is thus a simple *keyword* search, since it finds only those agencies that share terms with proposals. The relevant statistics for UKW, EC, and BF searches are shown in Table 1.

[4]Completely unconstrained BF search finds all agencies in the network, each by dozens of different paths, and requires hours of CPU time on a TI Explorer Lisp Machine. The data presented here are for a modified version of BF search that avoids nodes with extremely high fan-out and prunes paths longer than 4 links.

	UKW	EC	BF
fallout rate	64%	71%	94%
recall rate	44%	67%	100%
number of agencies found	164	406	2145
number of false positives	106	207	2013
number of hits	58	88	132
number correctly rejected	0	111	0

Table 1. Statistics from UKW, EC, and BF searches.

EC search has a higher recall than UKW and a lower fallout rate than BF. Its fallout rate is typically higher than UKW because it subsumes UKW: it finds all the agencies that UKW finds, then finds some more by exploiting semantic relations. Let us consider the utility of this additional search.

Of the agencies found by GRANT for the 27 test cases, the expert thought that 132 would be likely to fund their respective proposals. UKW found just 44% of these. To find the rest, it is necessary to exploit semantic relationships between the terms used in research proposals and agency descriptions. EC search found 67% of the agencies judged good by the expert. It found 242 more agencies than UKW search: 30 hits, 101 false positives, and 111 correctly rejected. So in the regions of the network that cannot be explored by keyword UKW search, EC search found 40% of the agencies it should, and incorrectly accepted 101 agencies, for a "marginal" fallout rate of 42% . In contrast, BF search found almost all the agencies judged good by the expert, but at a cost of a 94% fallout rate.

In practice, GRANT's mode of operation is EC search. It is preferred to UKW search because it finds more agencies, and to BF search because it has higher precision. BF search finds about 80 agencies per proposal at a precision of 6% — only 1 agency in 20 is truly worth pursuing. EC search reports fewer agencies (15 per proposal), has a better level of precision (29%) than BF search, and has an acceptable, intermediate recall rate (67%).

Since EC search subsumes UKW search, it also inherits a significant fallout rate. The fallout rate for agencies found by keyword UKW search is 64%, but the marginal rate for those agencies found by additional semantic matching is just 42%. Clearly, path endorsements can increase precision. But their utility is obscured to some extent by the fact that EC search "starts off" with the 106 false positives found by UKW search. With this proviso stated, we now explore how to increase the recall and precision of EC search.

Our experiments are designed to address two general hypotheses:

- GRANT's performance is due to its path endorsements.
- GRANT's performance is affected by the *structure* of its network, including the

lengths of pathways between proposals and agencies, and the degree of interconnection between nodes.

A third hypothesis is that GRANT's performance is affected by how its language of links is used to encode the interests of agencies. Since many people worked on GRANT's knowledge base, we were concerned that knowledge was encoded inconsistently. We calculated several statistics that measure consistency, but we did not find significant or even suggestive correlations of these measures with fallout rates. We cannot conclude that inconsistencies have no affect on GRANT's performance, because our measures of consistency may not be sufficiently sensitive. But we have found much stronger evidence for the other two hypotheses.

Structural Factors in Recall and Precision. We first calculated the recall and fallout rates as a function of the distance between proposals and agencies in EC search (Table 2). As noted, at distance = 2 EC has the same fallout rate as UKW search, which finds all agencies within two links of the proposal. Extending the search one more link increases the recall rate substantially (from 42% to 70%) and also raises the fallout rate somewhat. Interestingly, extending the search further has almost no affect on the recall rate but does increase the fallout rate. This suggests that endorsement-constrained search as implemented here offers most advantage when finding agencies based on a single semantic relationship between a term used in the proposal and a term used in the agency description. Increased fallout limits the utility of longer chains of relations.

length	fallout rate	recall rate
less than 3	64	42
less than 4	73	70
less than 5	78	69

Table 2. Recall and fallout rates for searches along pathways of different lengths.

The structural feature of GRANT's network that accounts for most variance in recall rate and fallout rate is the *branching factor* of nodes, that is, the number of links that connect nodes. In an experiment reported in Kjeldsen and Cohen (1987) we found that the fallout rate was correlated with the average branching factor of pathways to agencies. Average branching factor is the average of the number of links emanating from each node on a pathway. It is a measure of the "density" of the network in the vicinity of the pathway. We expected dense areas of the network to have low fallout rates relative to recall rates, since there are more nodes per agency in dense areas, and thus more basis for discriminating good agencies from bad ones. Table 3 shows the

percentage of the false positives found along pathways with low, medium, and high branching factors.

EC Search	average branching factor		
	2 - 7	8 - 15	> 16
% hits	20.3	40.6	39.1
% false positives	8.4	36.9	54.6

UKW Search	average branching factor		
	2 - 7	8 - 15	> 16
% hits	30.7	55.1	14.1
% false positives	8.4	37.3	51.8

Table 3. Hits and false positives for EC and UKW search, distributed by average branching factor.

Contrary to our expectations, the majority of false positives were associated not with low branching factors but rather with high ones. For EC search, 54% of the false positives were found on paths with an average branching factor greater than 16. For UKW search, 51% of the false positives were associated with high branching factor; furthermore, only 14% of the hits were found in these areas. We looked at the test cases individually to try to explain this result. Many of the false positives were associated with nodes with high fan-out, such as "animal" and "location." We believe that such nodes are relatively general, that their fan-out is due to their many specializations. To say an agency is associated with one of these general nodes is to say very little about its interests, so agencies found via these nodes are more likely to be false positives.

These data seem to suggest that we could increase GRANT's precision by pruning agencies associated with general nodes. In fact, this is an artifact of the way we calculate precision. We could certainly reduce the number of false positives this way, but we would also reduce the number of agencies GRANT finds, and so would have little effect on the fallout *rate*. Moreover, since the denominator of the recall rate is constant — the number of agencies judged good by the expert — pruning agencies can only reduce the recall rate. Clearly, false positives are associated with higher branching factors. However, the key to improving precision is not to prune agencies, but to restructure the network so that it has fewer pathways with high branching factors, that is, fewer nodes that represent very general concepts. For example, the current network defines *dandelion* and *tomato plant* as instances of the *plant* node, though they are obviously different kinds of plants. The distinction could be made by defining dandelion as an instance of a *weed* and tomato plant as a *domestic plant*, but because these nodes are not in the network, the fan-out of *plant* is higher than it should be and *dandelion* and *tomato plant* are not adequately discriminated.

The statistics in Table 3 suggest that the "ideal" branching factor is less than 16.

Another experiment was needed to pinpoint the ideal more precisely. Starting with the list of agencies found by the EC search and reported in Table 1, we ranked the agencies by their branching factors, and recalculated the recall rate and fallout rate for each successive level of the ranking. That is, we superimposed a ranking by branching factor on the list of agencies found by EC search and asked about the recall rate and fallout rate of all agencies that had, first, low branching factor, then those that had higher branching factor, and so on. (For reasons discussed below, we used the branching factor of the last node on a pathway instead of the average branching factor over all nodes on a pathway.) The results are shown in Table 4.

Agency is counted as "good" if the branching factor is:	fallout rate	recall rate	number of FPs	% change number of FPs	number of hits	% change number of hits
any number	73	63	219	2	82	1
16 or less	73	62	215	14	81	17
13 or less	73	53	188	55	69	15
10 or less	67	46	121	157	60	140
7 or less	66	19	47	81	25	25
3 or less	58	15	26		20	

Table 4. Fallout and recall rates from ranking agencies by branching factor.

These data suggest that disproportionate numbers of false positives are associated with low and moderately high branching factors. At the lowest level (branching factor of 3 or less) there are few false positives (26) and hits (20) because few nodes have such low branching factors. At the next level we consider agencies found via nodes with branching factor of 7 or less. 47 are false positives, an increase of 81%, and 25 are hits, an increase of 25%. Thus, fallout rate increases faster than recall rate for nodes with relatively low branching factors. When nodes with higher branching factors (10 or less) are considered, fallout rate increases by 157% and recall rate by a comparable 140%. However, adding agencies that are found by nodes at the next level of branching factor (13 or less) increases fallout rate by 55% but increases recall rate by only 15%. The rates then increase proportionately for higher levels of branching factor.

The greatest increase in recall and fallout occurs when we add the agencies found via nodes with branching factors between 8 and 10. Moreover, the numbers of hits and fallouts increase by roughly the same amount in this area (about 150%). In contrast, false positives increase more rapidly than hits at low (3 - 7) and moderately high (11

- 14) branching factors. This suggests that the "ideal" branching factor is between 8 and 10, and supports the hypothesis that recall and fallout rate are correlated with the generality – as measured by branching factor – of nodes. As mentioned above, we used the branching factor of the last node on a pathway – the one "nearest" to the agency and "furthest" from the proposal – to produce the data in Table 4. We reasoned that very specific nodes, those with low branching factor, would rarely be part of an agency description, and so would not be associated with many hits. On the other hand, as we argued above, nodes with very high branching factors are too general to represent the interests of an agency unambiguously, and so would be associated with high fallout rates.

The primary implication of these results is that knowledge engineers for GRANT-style systems should ensure that the definitions of new terms are as specific as possible. For example, the knowledge engineer should define a new plant in terms of the most specific possible subclass of plants, or perhaps create a new subclass, rather than linking the new plant to the general *plant* node. Currently, GRANT is programmed to avoid nodes with extremely high fan-out. An alternative would be to alert the knowledge engineer to them during the development of the knowledge base, to fix the problem before it arises. Then, any remaining nodes with high fan-out almost certainly denote concepts that are too general to be useful, and endorsements could be designed to avoid them, or to give them a low rank.

Endorsements as Factor in Recall and Precision. Our second hypothesis is that although the representation language for the network is probably sufficient to encode the meaning of research proposals and agency descriptions, these representations are not being exploited by endorsement-constrained search. Several findings support this hypothesis. In Kjeldsen and Cohen [15] we reported that just three path endorsements accounted for 85% of the hits but the same three led to 42% of the false positives. The culprits were:

- SUBJECT : SUBJECT-OF
- SUBJECT : SUBJECT-OF : SUBJECT-OF
- OBJECT : SUBJECT-OF

Despite the fact that 48 distinct relations are used in the network to connect concepts, just 3 (SUBJECT, OBJECT, and SUBJECT-OF) were sufficient to find the majority of hits and a sizeable portion of false-positives. This is partly due to the relative frequency of these links in the network: they are very common and so support a disproportionate number of path traversals. However, our data suggest that the reliance on these links is not due entirely to their frequency, and that intelligent use of other links could increase recall rate.

We measured the frequency with which different links were used to represent agency descriptions. These data are shown in Table 5. As expected, SUBJECT, OBJECT,

and FOCUS were most common, but WHO-FOR and LOCATION were not infrequent. However, these latter links were almost never traversed to find agencies: Table 6 shows the results of using the last link in a pathway (the one closest to the proposal) to rank the agencies found by EC search. If SUBJECT and OBJECT are the only links that GRANT is allowed to traverse, then it finds 74 hits and 179 false positives. It finds an additional 15 hits when it is also allowed to traverse FOCUS. But, remarkably, allowing it to traverse *any* link results in only 2 more hits: Most of GRANT's hits are found by following SUBJECT, OBJECT, and FOCUS links into an agency. Although WHO-FOR and LOCATION are used quite often to define the interests of agencies, they are not used to find the agencies. This is not surprising, since WHO-FOR and LOCATION are the final link in only 2 path endorsements. But it does suggest that using these and other links judiciously could increase GRANT's recall rate. In general, these results stress that path endorsements must reflect the conventions for representing concepts.

Link	Number of uses in agency definitions	Number of uses as last link of endorsements
subject	513	19
object	258	10
focus	238	17
who-for	124	2
location	80	0
dv	30	8
iv	20	5
rv	18	5

Table 5. Number of times each link is used to define agency interests, and number of times it is the final link in an endorsement.

Agency is counted as "good" if the last link in a pathway is:	fallout rate	recall rate	number of FPs	number of hits
SUBJECT or OBJECT	71	57	179	74
SUBJECT, OBJECT, or FOCUS	72	68	228	89
ANY LINK	73	70	251	91

Table 6. Fallout and recall rates from ranking agencies by final link.

To get a more complete picture of the utility of GRANT's path endorsements we would perform "ablation studies" — removing path endorsements one at a time to see how they affect recall and precision. Unfortunately, an exhaustive analysis of all endorsements would require weeks of computer time. Instead, we grouped the path endorsements and assessed the effects on performance of removing these classes. Every path endorsement is assigned to one of five classes that reflects the subjective probability that an agency found by that endorsement would fund the proposal. The classes are trash, unlikely, maybe, likely, and very-likely. We used these classes to rank as "good" or "bad" the agencies found by EC search, then recalculated recall and fallout rates for each rank. The results are shown in Table 7.

Agency is counted as "good" if it is found by an endorsement classified as:	fallout rate	recall rate	number of FPs	% change in number of FPs	number of hits	% change number of hits
very-likely	55	18	28	425%	23	78%
likely or very-likely	73	42	147	41%	54	59%
maybe, likely, or very-likely	71	67	207	4%	86	0%
unlikely, maybe, likely, or very-likely	72	67	216		86	

Table 7. Fallout and recall rates from ranking agencies by class of path endorsements.

When only *very-likely* endorsements are allowed, the numbers of hits and false positives are low (23 and 28, respectively). Adding in agencies that are found via paths with *likely* endorsements increases the number of false positives by over 400% to 147. This seems an excessive price to pay for the 78% increase (from 23 to 54) in the number of hits. In contrast, adding in agencies with *maybe* endorsements increases the number of hits by 59% and increases false positives by a significantly lower amount, 41% . (The main reason for the increase in recall is that FOCUS links are used in a preponderance of *maybe* endorsements, and are infrequently used in *likely* or *very-likely*. We saw in Table 5 that the FOCUS link is used frequently in defining agencies, and in Table 6 that inclusion of the FOCUS link increases GRANT's recall rate.)

Clearly, GRANT's fallout rate could be improved by refining its *likely* endorsements. The improvement in performance due to adding *maybe* endorsements — specifically those dealing with FOCUS links — convinces us that it is possible to add endorsements that will increase recall and precision simultaneously. Table 5 suggests that these endorsements should exploit WHO-FOR and LOCATION links, which are used to define agencies but are rarely traversed to find them. We are currently designing new endorsements, though they will have to be tested on a new set of proposals to ensure

that they are not simply "tuned" to the current test cases.

4 Lessons learned

Now we would like to take a step back from the details of the testing and summarize what general lessons we have learned. Most importantly we have shown the potential of constrained spreading activation and semantic matching. In the information retrieval task which Grant performs, the addition of semantic information to a straight keyword search was able to improve performance significantly. In Cohen (1987b) we have identified some characteristics of tasks that we feel Grant-like systems may perform well. Here we will discuss what can be done to further improve the performance of such systems. Following that we will look at some of the theoretical lessons we have learned.

Knowledge Representation This testing and our attempts to fine tune the rule set, have pointed out that a major cause of trouble is the representation of research interests. In the development of any similar system, this should be a primary focus of effort. The representation language must be sufficiently powerful to extract the differences between alternatives and clear enough that the knowledge engineer can be consistent in its use.

Bruce Mccandless of the ORA has come up with a new case semantics for representing research interests in Grant. Rather than being based on a SUBJECT/OBJECT/FOCUS description of the research itself, this is aimed at describing the product of the research. The product may be a physical object or a better understanding of some process. It is generally intended to be used by some audience, with some specific purpose in mind. Thus the new representation language has well defined slots for each of these features. Grant's funding source knowledge base is in the process of being updated to make consistent use of this new scheme.

The rest of Grant's knowledge base, the network of relations between terms, seems to be reasonably good for its task of finding semantic matches. It has about the right level of detail and a reasonably good representation language. Inconsistencies in the network base do not appear to be a major factor in the performance of the system. In general, Grant is an interesting demonstration of what can be accomplished with a broad, shallow, and somewhat inconsistent knowledge base.

Some changes to the knowledge base may be warranted however. As our branching factor tests discovered, an increased use of subclassifications may improve Grant's FPR. Another advantage of an increased use of subclassification is that the system may be able to make use of longer chains of inference. As we have shown, most of Grant's useful work is done using relatively short paths. Rules leading to longer paths tend to increase the number of false positives without helping find hits. This keeps Grant from finding useful though more tenuous relationships. Part of the problem may be that

long chains of inference very quickly reach a node with high fan-out, that is a node representing a classification which should be subdivided. From this node links exit to areas of the network which are conceptually unrelated to the starting point. There is an idea here of *locality of semantics*, that is within an area of the network, concepts tend to be related by close associations. Overly general nodes lead very quickly out of the local semantics and so limit the length of useful paths. With a knowledge base having more precision, that is more subclassifications and longer chains of links to reach the top of the net, it may be possible to make use of longer chains of inference.

Rules As has been mentioned, Grant has rules to help it avoid general nodes, that is nodes which are thought to be to near the top of the concept hierarchy. This is a very important factor in the performance of the current system. This importance may indicate that the path grammars need to be modified. A path endorsement which avoids expanding a general node is essentially a path endorsement which says any number of links followed by node X followed by any number of links is a bad path. The general node can be thought of as the *context* in which the path is used. Path endorsements, however, are based on the assumption that relationships between topics are sufficient to encode semantically meaningful associations *regardless* of the context in which they are used. For example, a component of a social group (a person) may not interest a funding source supporting investigation of interactions within such groups, while a component of a mechanism may well interest an agency willing to fund improvement of that mechanism. Thus the SUBJECT:HAS-COMPONENT:SUBJECT-OF path is good when the SUBJECT is a rotary engine, but not when the SUBJECT is a soccer team. A rich enough set of links could capture such subtle differences (it could use HAS-MEMBER when referring to a social group) but this may lead to an unmanageable proliferation of links. Another alternative is to include in the rules a notion of context so that we can tell Grant to use the above rule only when the SUBJECT node is connected by one or more ISA links to "thing." Addition of contextual information to path grammars will allow Grant to make use of more than just links in its evaluation of a node and may improve its power to discriminate good matches from bad. The importance of general nodes in the current implementation argues that such contextual information may be needed for the system to reach optimum performance. Grant currently has the ability to create such grammars but they have been seldom used.

Knowledge Engineering Among the lessons we have learned is the very practical one that powerful development tools are needed to ensure accurate construction of such a system. In Grant there are two areas where such tools are needed; building the semantic network and designing a rule set.

For building the knowledge base the most important tool is one which would provide accurate, flexible display of the nodes of the knowledge base. A graphical display of the nodes in the vicinity of those being worked on would be the best alternative. Current

tools only allow individual nodes to be shown. Being able to see several node definitions on the screen would provide many advantages. It would help avoid inconsistency in link use by providing immediate examples, it would insure consistent semantics within the local area of the network, and would serve as a visual reminder of what remains to be done. An example of the utility of a graphical display came when we ported Grant's knowledge base into Intellicorp's KEE. KEE has graphic display facilities capable of showing only the ISA/EXAMPLE links in Grant's network. Even this limited display of the net showed us many inconsistencies in the way the nodes had been coded, and provided insights into how it could have been done better.

Some problems with the knowledge base can be easily identified. Nodes with high fan-out reduce performance and should be avoided. Improper link use can sometimes be identified. A tool which would keep track of information such as this and alert the knowledge engineer to potential problems would help the situation. One possible implementation is to create a small rule based system which watches over the shoulder of the user. The rules would embody what we have learned about good representation practices, and point out potential problems such as high fan-out.

Probably the most important set of tools are those which help design a rule set. We found that we could not live without two tools; one to find all paths (given a distance limit) between two nodes, and one to find what rule had lead to a particular agency after a search was completed. The former needs only to search the knowledge base, the later must peer into Grant's knowledge structures. These were invaluable in designing the very specific rules needed to improve the performance of the system.

The process of creating a rule set may be able to be completely automated. While adjusting the rule set, we found ourselves performing an algorithm very similar to the one used by the learning system described in Appendix 2. The algorithm iterates through many example cases, creating and adjusting the ranking of rules. It finds paths to positive and negative examples provided by our expert and either creates a rule or adjusts the ranking of an existing rule to lead to that example, or avoid it. Our experience in manually adjusting the rule set argues for the utility and potential success of such an approach.

Matching Among the reasons Grant was developed was to explore the utility of semantic matching in uncertain domains. Experiments with Grant have shown the importance of semantic matching. However one of the most prominent lessons from our experience is that we have rediscovered that single associations are not always enough. Unfortunately previous solutions to this problem, such as Tversky's contrast model are limited by their reliance on syntactic matching. Combination of the two may give a powerful matching algorithm. Semantic matching, as implemented here, appears to be good at pruning large numbers of bad matches. It is not so good at evaluating a degree of match at a more detailed level. For that we need a more powerful matching algorithm, which in turn would get overwhelmed trying to determine degree

of match between a large number of possibilities. We see a symbiosis of these two working something like this: Spreading activation from a proposal prunes the possibly matching agencies to the best dozen or so. It hands these to a matcher, along with a ranked list of the semantic associations between the proposal and each agency. Now assume each research topic description has its slots labeled as to which must match and ranked as to which are most important to match. The matcher discards those whose "must match" slots don't or whose market factors rule out a match with this proposal, and ranks the rest by combining the slot rankings and the rankings on the semantic associations between slots. Obviously work must be done to determine a useful combining function.

The real win here will not come from a tightly constrained spreading activation search, rather from a loosely constrained search and intelligent use of the results. By combining a simple full matching algorithm and an underconstrained spreading activation search, neither of which performs very well, we may get a powerful retrieval mechanism.

Uncertainty One of our goals with Grant was to integrate management of uncertainty into the reasoning process. Cohen (1983) has described what he calls the parallel certainty inference, where an inference is performed as though it were truth preserving while a separate measure of our certainty in that inference is maintained. This is the method of handling uncertainty used by most systems. Grant was constructed to work in an uncertain domain without this explicit and separate representation. Unfortunately, the way the system was implemented this doesn't entirely hold. We still have an inference and a separate measure of confidence in it. The difference is that the measure of confidence is computed as needed rather than maintained in parallel.

This is not to say that Grant does not contribute to the study of reasoning under uncertainty. It has become clear lately (Cohen, 1987a) that a powerful method of managing uncertainty is to use knowledge about uncertainty to control the actions of the system. Grant relies on its measure of confidence almost exclusively to guide the reasoning process. It knows what conclusions it has drawn in which it has confidence, and continues on from those conclusions until it finds its goal.

However, like most systems, Grant's knowledge of its ignorance is limited to a course measure of its certainty in a conclusion. It may be able to benefit from a specific representation of what it does not know. Given something more detailed than a simple ranking of a path, such as in what conditions a certain path is useful it may more accurately judge its level of certainty in a conclusion and be able to search more effectively. Given knowledge about why an inference does or does not hold may help it explain its decisions. Unfortunately this is an area where Grant is no better than its peers.

Inference Could a system such as Grant ever achieve perfect performance? It may be that an absolute minimum FPR is determined by the approach itself. It can be argued that the basic mode of inference used in Grant is abduction and that abduction can lead to false positive results.

Our goal is to find concepts that match our starting point by some measure. We take as an axiom that matching concepts are likely to be semantically related. In Grant, this is the premise of the abductive inference that the existence of semantic relationships between concepts implies a match between them. That is, given the premise that matching concepts are related, if Grant finds a relationship between concepts, it infers that the concepts match. Because this is an abductive inference, not a deductive one, it will occasionally be wrong. When it is it will lead to false positive results, as Grant will find semantically related concepts that do not match.

An extension of our axiom would state that the better two concepts match, the more relationships they are likely to share. Using an abductive argument once again we can conclude that two concepts with several semantic relationships between them are likely to be a good match, and at least are more likely to be a good match than concepts with a single relationship. Thus one way to reduce false positives due to Grant's method of inference is with a matching algorithm which takes into account multiple associations between concepts, such as in the one proposed above.

5 Future Work

Over the last years we have built and expanded a system, run an extensive series of tests on it and started the motions of getting it into use. What remains to be done? There are several aspects of Grant which make it a prime candidate for further work. We have here a system with a reliable control structure and a large semantic network representing many person hours of knowledge engineering. The methods Grant uses are relatively simple and easily understood, but the implementation is flexible enough to support much more complex methods of controlling spreading activation. Moreover, we understand rather well how and why the existing system works. This provides a basis by which to judge the effect of changes to the system. There are five areas which come to mind as viable, near-term projects.

Improve the knowledge base The existing knowledge base is a very broad but shallow cross section of knowledge. As we have stated many times, it has been designed and built with Grant's performance task in mind. In our opinion, however, it is general enough that it can be used for other tasks without major modification. Unfortunately we have shown that it has some problems with consistency and precision. We have learned a great deal about what is needed in such a knowledge base for it to be used reliably. With a reasonable amount of effort Grant's knowledge base could be made

much more usable. Specifically what needs to be done are two tasks; improve the consistency with which links in the net are used and lower the average branching factor by adding subclassifications of nodes with high fan-out. Neither of these tasks is very difficult and the payoff is that it would make the knowledge base a valuable resource for further work.

Add Full matching In our minds the most interesting open project is to integrate Grant with a more complete matching algorithm, such as the one described above. Algorithms which compare frames for shared properties can benefit from the use of semantic associations while any Grant-like system would benefit from a more complete matching algorithm. The trick will be in how to combine differently ranked semantic associations into a measure of the degree of match.

Explore utility of context in path grammars There are many improvements which can be made to the current system which would provide interesting projects. One is to explore the utility of context in path grammars. Is it possible to improve the performance of the system if contextual information is added to the rule set? Is there a better way to control spreading activation to find potential matches? Carefully controlled experiments can be run now that we have a performance baseline.

Learning Grant also seems to be an excellent test bed to try learning and knowledge acquisition algorithms. There is a reasonably well defined performance task and evaluation criteria, there is a large initial knowledge base from which to start, examples of good behavior (correct matches) are easily generated and there are two aspects of the system which could benefit from learning. Improving the network is one. There are too many interrelations for a knowledge engineer to keep on top of. A good knowledge acquisition interface and/or a learning system which looked over the shoulder of the programmer to watch for inconsistencies and missing links would be invaluable. Improving the rule set could also benefit from learning. An attempt was made to learn rules from examples is described in Appendix 1. The algorithm used was simple, but could be easily extended to include induction, analogy and perhaps other tools.

Abstraction of Grant Finally there are the cognitive science issues which would be interesting to explore. Some work was started on describing Grant in terms of plausible inference, that is formalizing the inferences it is making. The relationship between the techniques Grant uses and the concepts of representativeness and availability has never been followed up on. Continuation of these projects would be valuable to further our understanding of the broad but shallow reasoning Grant does.

6 Conclusion

Through our work with Grant we have shown the potential of the techniques it uses. Grant's greatest asset is its ability to find potentially matching entities using semantic information and identifying the associations between them. In a domain such as information retrieval, where a high FPR is acceptable, Grant's existing techniques may be sufficient. In a more stringent environment, however, changes may have to be made. We have identified improvements such as better full matching, addition of context, and more careful knowledge engineering (including an improved set of links and better enforcement of network semantics) which a Grant-like system could use to improve its performance. Finally we have tried to identify what we have learned from our experience with respect to matching and the management of uncertainty.

7 Extensions to GRANT

Grant led to two projects that are designed to improve and generalize inference by constrained spreading activation. These are discussed next.

Spreading activation search of a semantic net can be used as a search for representative concepts. A method is put forward for learning rules to constrain such a search in a general net. This method uses an iterative parameter adjustment procedure, based on a comparison with supervised learning pattern classification tasks, to find paths through a net which lead to truly representative concepts using examples provided by a human tutor.

Introduction

Semantic nets can be a convenient way to represent the relationships between concepts. One application of such nets has been to use them to find topics related to, or representative of some starting point(s) (Cohen, 1985). Given a net representing general relationships, traversal rules can be crafted, which constrain a spreading activation search of the net to lead to concepts which are representative of the start ng point by some measure.

Unfortunately, it is not easy to generate these traversal rules. To do so requires both knowledge about reasoning in a particular domain and knowledge of the structure of the net itself. Even a domain expert may not be consciously aware of her own reasoning process, or may be unable to put it into useful terms. During the development of Grant,[5] traversal rules were hand generated using a witches brew of decision examples, intuition, trial and error and luck. On the other hand, it was easy for our expert to

[5]Grant is a system which finds funding sources for research proposals using the described techniques (Stanhope, 1986).

provide examples of correct decisions given some context for them. An automated method to generate traversal rules using such examples would facilitate knowledge acquisition in any similar system.

At any given time, in a spreading activation search, we may extend the search by expanding one of the nodes on the frontier of the search, adding all nodes which are directly related to that node to the frontier. The assumption in Grant was that the best node to expand was the one most representative of the starting point of the search, and that this degree of fit was indicated by the sequence of associations (links) traveled from the start node.

Thus a convenient (if large) set of potential traversal rules for our learning system is the set of all combinations of links in the network. If we can use the path to a node to find a weight for each node on the frontier indicating how well it supports associations of the type in which we are interested, then at any given point in a spreading activation search, the node who's path has the highest weight is expanded, and at each step we are pursuing the most representative concept available. The problem becomes how to assign weights to the paths.

Learning in this context seems to have similarities to a Supervised Learning Pattern Classification task as described by Barto and Anandan (1984). It therefore seems to make sense to try to find the weights using methods found to be successful at such a task. In supervised learning pattern classification, a system learns to classify input states into one of several categories. The training set is a set of states x, each paired with it's correct classification, y. The learning system adjusts it's decision rules in order to increase the probability of classifying x correctly. X is generally a vector, and can be thought of as a set of features describing a state. Each of these features has an associated weight. The weighted sum of the input vector is used to classify that state. For example, in a two class system, a threshold on the weighted sum is used to classify the input. During training an input vector x is supplied to the inputs. The weight vector is adjusted using the correct classification, y. Generality occurs when some x is input which has not been seen before. A supervised learning pattern classification system will attempt to classify it into the class who's members it matches best.

In grant an input state is a description of one node at the frontier of a spreading activation search. One set of features we can use to describe this state is the path followed to that node, and the context of that path. Context refers to the nodes passed thru along the way. These are the features available to the system during the search. Our output classification is Good or Bad, but rather than use a threshold, we will output the weighted sum of the input features as a measure of the 'goodness' of that state. The training set we have mentioned can be thought of as a set of descriptions of good, or possibly bad states, using another set of features, in this case the representative node itself. A training pair (A B) says that for a spreading activation search starting at node A, B is a representative concept.

If we can have Grant translate these examples into the path/context representa-

tion, we can use an iterative parameter adjustment procedure similar to those used in supervised learning pattern classification systems to find the weights for this weighted sum.

Implementation

Let each possible path in the net have a weight vector associated with it, where one of the weights is the weight of the path itself, and the others are grouped as follows. There is a group of weights for the start node of the search, the end node, and each node passed thru. In each group there is one weight for each type of node in the net[6]. Thus the weighted sum which determines a state's 'goodness' is the sum of the path weight, and one weight from each group, chosen by the node types encountered along the path. During spreading activation, the system will compute such a weighted sum for each node on the frontier and expand the node with the highest weight.

We will attempt to have the system learn the weight parameters which will generate weighted sums for paths which in turn will lead the system to the associations in the training set. Once adjusted, the weights will have generality in the sense that they will apply to situations not in the training set. This will occur because we are generating weights for semantic paths between concepts and adjusting those weights according to the context of the path. This process is independent of the end points of the search, represented by the training set.

The first step is to translate a training pair to the path/context representation. The system performs a breadth first search starting at the start node of a training pair, to find the shortest path to the end node. This is assumed to be the correct path, that is the path the expert traveled in her decision making process. The path and it's context are recorded. This is done for all pairs in the training set. Next the system searches from each start node using it's current weight vectors [7]. It halts when it has found the end node, or when it has found a path longer than the correct path. If it has found the end node, it proceeds to the next training pair. Otherwise it determines the node on the search frontier that is farthest along the correct path, the node which should have been expanded to continue along the path to the end node. The path to this node is the longest discovered subpath of the correct path. For the rest of the nodes on the frontier, the active weights of their paths are decremented by a small amount. The weight of the longest subpath is incremented by an amount equal to the total amount removed from the other nodes.

[6] There are several ways to define the type of a node. One example is to say that a node, Y, is of type X if the node X can be reached by following a chain of ISA links from Y.

[7] If a node has not been encountered before, it receives a random weight during the weighted learning search in order to eliminate the breadth first search which occurs the first time thru the learning loop, when no path has a weight. When the system is in use, such nodes are given a weight of 0.

Status

In order to test the feasibility of this learning scheme, a simplified version of the algorithm has been implemented on top of the existing Grant. For this purpose, only the weight of the paths themselves are used. It is expected that this will be able to learn useable traversal rules, but that later addition of context information will fine tune the rules, and improve performance. Some testing has been done on this implementation, though by no means enough.

Evaluation Criteria

There are at least two ways to test a program such as this. The first, objective testing, determines if the learning algorithm itself works. In other words does the system find the nodes in the training set sooner after training than it did before. The second, subjective testing, attempts to determine if the system will work in the real world. Here we test to see that after the system is trained on one set of examples, it finds other meaningful associations which were not in the training set. This is actually a test of the validity of this type of generalization.

Results

For the objective testing, a training set was created by selecting several start nodes, and giving them to the original Grant system. Each was used as the start node of a search using the old (hand crafted) traversal rules. Several training pairs were created from each start node by combining it with the nodes which Grant found along strong[8] paths. These pairs ranged from 1 to 4 links apart. More than half had a path length of 2 with most of the rest split between 1 and 3.

Using the new Grant, a sample of these pairs were searched for before any learning was done. In this configuration all of the weights were 0, creating a breadth first search. For a typical pair, over 100 nodes were searched before finding the end point. After one pass over the training set, there was a noticeable improvement in the number of nodes searched. After 4 passes, typically less than 10 nodes were searched before finding the goal. Most of those were from pairs also in the training set.

This indicates that the system is learning to follow paths leading to the training pairs, however the testing is obviously incomplete. A more complete analysis of these test results is in order, as well as testing with several different training sets. No attempt has yet been made at subjective testing.

[8] Traversal rules in the original Grant put paths into several classes, the best of which was strong.

Discussion

The weight adjustment procedure solves one problem which had plagued the original Grant. This is a problem similar to the horizon effect in game tree search. No attention was given, when creating traversal rules, to the paths leading to those in the traversal rules. Thus Grant at times expanded the best paths available at each step till it had reached one of it's search limits, maximum search depth for instance. There was no way of knowing that if it had followed a lower ranking path, it would have reached a high ranking path. In other words, highly ranked paths could be hidden behind lower ranked subpaths. The new system avoids this problem because it tends to reward paths it finds *leading* to representative concepts, rather than the path to the concept itself. This may lead to a somewhat higher false positive rate, as short paths which may or may not be good associations, lead to strong longer paths, and so are rewarded. We expect, however, that this will be more than offset by the empirical accuracy of the rules.

Another problem for the original Grant, was when if what constituted a good association changed from one part of the net to another. If a SETTING-ISA path is good for illnesses, but not so good for living things, the system got into trouble when it tried to use SETTING-ISA paths everywhere. We hope to show that this is solved by use of the context information in the weight algorithm.

Further Work

There are several areas open for further work. First and most important is further objective testing to see if the learning algorithm can produce useable traversal rules. Next some attempt must be made at subjective testing to test the validity of using paths for this type of reasoning. When we have evaluated the performance of the system using just path weights, the next goal will be to add the context information to the algorithm, and assess it's effects.

Possible Enhancements

A more sophisticated form of generalization could be added. One technique which was useful in the original version of Grant was the use of wildcards in path descriptions. It may be possible for the system to notice, for example, that there were many heavily weighted paths starting with a cause link. A new rule for 'cause-*' (cause followed by any links) could be created and allowed to compete for weight with the other rules. This would give the system an explicit form of generalization.

Notes

One thing we noticed while working with the system was that there was something to be learned from looking at the training set after it was compiled into paths. First we were easily able to recognize when the system had connected two nodes by a poor path. It would have been easy, if our goal was to maximize performance, to fine tune the training set at this point. More importantly, sometimes those bad paths pointed out missing links in the net itself. Allowing a user access to the compiled training set could be important for a functioning system by showing her where the net need be improved. Better still, perhaps there is a way for the system to recognize these missing links itself. The system could look for long paths which connected pairs in a training set as a guide to where to add links in the net. If it sees a suggested link several times, it could add a new link there. Along with this may be needed a mechanism to eliminate links to avoid a link explosion.

Another way of viewing this learning algorithm is to think of the system as shifting weight from the net in general towards paths which lead to the associations in the training set. As the sub-paths of the training path are learned, the system learns to follow paths leading to the associations in the training set. This has similarities to Holland's Bucket Brigade algorithm, where the desired action receives a reward. It in turn rewards actions which lead to it, and so on, ending up with the actions leading to the desired ends occurring with a higher probability. It may be useful to pursue this similarity further.

Conclusion

The learning system, as implemented so far, shows some promise. It's advantages seem to be 1) creation of traversal rules from empirical data gathered from a domain expert, rather than subjective knowledge engineering. We hope to show that this leads to more accurate, and so useful, traversal rules. 2) Generality from some small set of training pairs to the whole net. 3) A knowledge engineering interface closer to the domain, asking the expert to think in terms of representative concepts and correct decisions, rather than chains of associations.

8 Plausible Inference

This research is concerned with the formal underpinnings of common sense plausible inference, the ability to give plausible answers to arbitrary questions from a very large knowledge base of associated statements. The goal is to find one or more answers to a question by consulting the knowledge base, and to say which of the answers are most credible. This has been a goal of AI since its earliest days (McCarthy, 1958, 1968), and is now seeing a resurgence (Collins, 1978a,b; Lenat et al, 1986). The motivation for

such work comes from the increasing realization that powerful AI programs will depend on very large knowledge bases. It will be necessary for the system to use the knowledge base to answer questions that were not anticipated at the time of its construction. To handle both the broad ranging nature of possible queries, and to make use of large amounts of knowledge in an efficient manner, it is expected that the use of heuristics, or plausible inference rules, as well as traditional truth-preserving ones, will be necessary. Our research is directed by these concerns, as well as by a desire to bring a formalism to plausible reasoning similar to that enjoyed by deductive logic, so that systems using plausible reasoning need not have their semantics established on a case-by-case, ad hoc basis.

The most important question to be answered about plausible inference is how to judge its credibility. Since plausible inference need not be truth-preserving, some other semantic property besides truth must be the basis of judgments of credibility. We propose to develop a semantics for common sense plausible inference based on the associations that hold between the antecedents and consequents of inferences. Our approach is strongly motivated by evidence-based control: the credibility of a statement is represented by reasons *why* it may be false, reasons that can be used to control backtracking and retraction of plausible but false inferences.

Plausible inferences, unlike deductive inferences, need not be truth-preserving. The distinction is clear in a contrast between two rules of inference, modus ponens and abduction:

Modus ponens is truth-preserving: if A $\rightarrow$ B and A are true, B cannot be false. Abduction is a rule of plausible inference because A is a plausible conclusion given A $\rightarrow$ B and B, but this conclusion is not *guaranteed* to be true, as the conclusion B is in modus ponens.

Since rules of plausible inference do not make guarantees about the truth values of their conclusions, how are we to assess the *credibility* of conclusions of plausible inference? In the deductive case we associate credibility with the semantic property *truth*: true statements are credible, false statements are not. What semantic property of conclusions derived by plausible inference will be associated with credibility? We could use truth, since some conclusions of plausible inference have truth values. The problem is that rules of plausible inference make no guarantees about these truth values, as rules of deductive inference do. So the question remains: What properties of conclusions are preserved by rules of plausible inference and are the basis for judgments of credibility?

Truth is not the semantic property we seek to preserve in plausible inference. This is because of our abiding interest in uncertainty, the state of not knowing whether a proposition is true or false. Many attempts have been made to modify deductive logic to represent uncertainty, including modal logics, 3-valued logics, nonmonotonic logics, fuzzy logics, and probabilistic logic (Turner, 1984; Zadeh, 1975; Nilsson, 1984) Some of these approaches "sequester" uncertainty by introducing a new argument that represents the uncertainty but is itself true or false. Modal logics do this. Other

approaches augment the values true and false; for example, three-valued logics add the value "unknown," and fuzzy logics introduce numeric arguments. Nonmonotonic logics go further and replace the notion of truth with one of *support*. Nonmonotonic formulations differ; in McDermott and Doyle's version, the notion of truth is generalized to *support* and falsity to lack of support (McDermott and Doyle, 1980).

Although uncertain statements are neither true nor false one can say a great deal more about them. Extensions to logic, however, say little. With the possible exception of nonmonotonic logic and dependency-directed backtracking, none of the extensions to logic enable us to say why we are uncertain and what we might do about it (de Kleer, et al, 1977). Shortly, we will discuss an alternative approach, but first we must address another common paradigm in AI for plausible inference and explain why we are avoiding it.

Much of the AI community favors probabilistic representations of uncertainty. We believe that, with one exception, the semantics of these representations are opaque. The exception is when the probabilities are relative frequencies, combined by Bayes' theorem. This case is akin to deductive inference in that a semantic property (relative frequency) is guaranteed to be preserved by a rule of inference (Bayes' theorem). Just as we associated credibility with truth in deductive inference, we can associate it with relative frequency in probabilistic inference. In both cases, we can guarantee that the credibility of a conclusion can be unambiguously determined. Unfortunately, the numbers used in knowledge systems are not relative frequencies. Until we know what they represent, we cannot know whether their intent or meaning is preserved by the functions that are used to combine them. The plethora of combining functions discussed in the AI literature suggests that no common interpretation of degrees of belief is available (Duda and Hart, 1976; Pearl, 1982; Shafer, 1976).

So we are led back to the question, if truth or relative frequency are not the basis of credibility when reasoning under uncertainty, what is? What properties of statements determine their credibility, and can we guarantee that these properties are preserved by inference rules? In Section 4 we saw that the credibility of inferences depends on the semantic associations on which they are based. For example, if a researcher is interested in VLSI layout, and a funding agency is interested in electronics, the fit between them is good and the agency is apt to fund the proposal. The semantic association between electronics and VLSI is "has-subfield," and it is the basis of this plausible inference:

$$\frac{\begin{array}{c}\text{interested-in(agency, electronics)} \\ \text{has-subfield (electronics, VLSI)}\end{array}}{\text{interested-in(agency, VLSI)}}$$

In brief, degree of fit between two objects, X and Y, was defined to mean that some rule of plausible inference could be invoked to conclude interested-in(agency, Y) given interested-in(agency, X).

The GRANT system (Section 4) sets the stage for the current research. It is the first step toward a common sense plausible inference system as defined above – a program that answers arbitrary questions from a large, associative knowledge base. But GRANT does not, in fact, answer arbitrary questions. It answers the single question, "If a funding agency is interested in X, will it be interested in Y?" It can be generalized to a common sense plausible inference system as follows:

1. Assume that all questions are about *properties* of objects; for example, "Does Fido have fur," or "Is coughing caused-by bronchitis." Abbreviate such questions $R(O_1,O_2)$?; for example, caused-by(coughing,bronchitis)?.

2. The answer to $R(O_1,O_2)$? is yes if the knowledge base contains O_1 and O_2 connected by R. The answer is plausible if there is a rule of plausible inference of the form

$$\frac{\begin{array}{c}Q(O_3,O_2)?\\ R(O_3,O_1)\end{array}}{R(O_1,O_2)}$$

and $Q(O_3, O_2)$? is plausible. For example, imagine asking a system, "Are gin-and-tonics intoxicating?" or, has-effect(gin-and-tonic, intoxication)? Assume that the objects gin-and-tonic and intoxication are *not* linked by has-effect in the knowledge base. The question can be answered, however, by plausible inference using the rule

$$\frac{\begin{array}{c}\text{has-component}(x,y)?\\ \text{has-effect}(y,z)\end{array}}{\text{has-effect}(x,z)}$$

and the knowledge that gin-and-tonics contain alcohol and alcohol is intoxicating:

$$\frac{\begin{array}{c}\text{has-component(gin-and-tonic,alcohol)?}\\ \text{has-effect(alcohol,intoxication)}\end{array}}{\text{has-effect(gin-and-tonic,intoxication)}}$$

Property inheritance in frame systems is a special case of this kind of inference. The rule for property inheritance is

$$\frac{\begin{array}{c}\text{isa}(X,Y)\\ R(Y,Z)\end{array}}{R(X,Z)}$$

where R is any relation. For example, isa(collie,dog) and part-of(dog,fur) implies part-of(collie,fur). The approach we propose here allows us to infer the answers to questions based on semantic associations other than isa. Thus, the approach unifies several kinds of plausible inference, including causal inference (Weiss et al, 1977).

The model of plausible inference is not complete, however, since it lacks statements about the credibility of inferences drawn by plausible inference rules. Obviously, we do not intend to include rules that draw erroneous conclusions, but credibility is not guaranteed, as it is in logic, by plausible inference. We discussed how our rules implement a notion of credibility based on degree of fit, but this still does not guarantee credibility. We know of two general approaches to this problem. One is to attach to each conclusion a set of conditions that, if met, would increase its credibility. Collins, who developed this idea, calls these *certainty conditions* (Collins, 1978b). The other is to attach a set of conditions that, if met, would decrease credibility. We have called these *negative endorsements* (Cohen, 1984). From the standpoint of control, certainty conditions can guide a system to increase its belief and negative endorsements can help a system recover from errorful conclusions by pointing to reasons a conclusion might be wrong. Obviously, both are required for evidence-based control.

Given a set of rules of plausible inference, with reasons to believe and disbelieve their conclusions, we can engage in a range of common sense plausible inference tasks. Our proposed work thus involves several stages:

- Develop common sense plausible inference rules. These are based on semantic associations, so clearly we need a set of associations at the outset. We began with the associations in GRANT's knowledge base. Next, we generated all combinations of associations of the form

$$\frac{\begin{array}{l}A_1(x,y)\\ A_2(y,z)\end{array}}{A_1(x,z)}$$

  These can be filtered by case-semantic considerations: y must be a particular *kind* of object to fill the A_1 case of x, and z is also restricted by its relation to y. In many cases, though, z will not fill the A_1 case of x, and so a potential rule can be filtered out. Even with this filtering, GRANT's associations generated about 600 rules of plausible inference.

  The rules are further pruned by automatically generating, from GRANT's knowledge base, examples of inferences made by the rules. Thus we can select empirically a set of rules that make a high proportion of truly plausible inferences.

- Endorse the rules. Given these rules it remains to specify the conditions under which they are more or less likely to generate plausible conclusions. This work remains to be done.

- Test the rules. Recently, Cohen et al. (1985) tested GRANT by comparing its performance against that of an expert. The same approach will be used to test our common sense plausible inference system both in the GRANT domain, for which we have a very large associative knowledge base, and in other associative domains such as causal reasoning.

Further extensions involve generalizing rules of plausible inference to include conjunctions, negations, and quantification. It will probably be easy to make these extensions given the propositional form of the rules as shown above. However, the inference mechanism that underlies GRANT is a tightly-controlled spreading activation. This has several advantages that are discussed in Cohen et al. (1985), so we want to maintain this approach in our proposed work. We currently know how to model the plausible inference rules above as spreading activation, but we are not sure how to extend this approach when the rules include conjunctions, negations, and quantifiers.

The result of this work will be a set of rules of inference whose plausibility for the GRANT knowledge base has been discovered empirically and confirmed by comparison with expert judgment. We hope, however, to go beyond this result to explore the reasons WHY the rules discovered are plausible, in what situations they would not be plausible, etc. To this end, we plan to extend our work on plausible reasoning to domains that already have algorithmic solutions (e.g. deadlock prevention in operating systems). The use of an algorithmic solution as a foil for plausible ones will aid in the discovery of formal characterizations of the nature of plausible inference rules.

Chapter 3

Prospective Reasoning

1 Introduction

MUM is a knowledge-based consultation system designed to manage the uncertainty inherent in medical diagnosis (the acronym stands for Management of Uncertainty in Medicine). Managing uncertainty means planning actions to minimize uncertainty or its consequences. Thus it is a control problem – an issue for the component of a knowledge system that decides how to proceed from an uncertain state of a problem. Uncertainty can be managed by many strategies, depending on the kind of problem one is trying to solve. These include asking for evidence, hedging one's bets, deciding arbitrarily and backtracking on failure, diversification or risk-sharing, and worst-case analysis. The facility with which a consultation system such as MUM manages uncertainty is evident in the questions it asks: it should ask all necessary questions, no unnecessary questions, and it should ask its questions in the right order. These conditions, especially the last one, preclude uniform and inflexible control strategies. They prompted the development of the MUM architecture in which control decisions are taken by reasoning about features of evidence and sources of uncertainty.

1.1 The Goals of MUM

MUM diagnoses diseases that manifest as chest pain and abdominal pain. This includes taking a history, asking for physical findings, ordering tests, and prescribing trial therapy. Physicians call a diagnostic sequence of questions and tests a *workup*. MUM's *primary* goal is to generate workups for chest and abdominal diseases that include, in the correct order, all necessary questions and tests and none that are superfluous. Since we built MUM to study the management of uncertainty, the goal of correct diagnosis is secondary to generating the correct workup. We were influenced by a distinction physicians make between *retrospective* diagnosis, in which all evidence is known in advance and the goal is to make a correct diagnosis, and *prospective diagnosis*, which

emphasizes the workup and proper management of the patient, even under uncertainty about his or her condition. MUM is definitely prospective. Figure 3.1 illustrates part of the workup for coronary artery disease. Clearly, we could build a system that follows this and other stored workups, but the point of the research is to be able to reason about the features of evidence, and the uncertainty in partially-developed diagnoses, to decide which questions to ask next. If MUM does this properly then its questioning will correspond with a standard workup, or at least be a reasonable alternative workup.

1.2 Managing Uncertainty and Control

MUM is based on the idea that managing uncertainty and controlling a complex knowledge system are manifestations of a single task, namely, acquiring evidence and using it to solve problems. There would be little basis for variation in problem-solving strategies if all evidence was equally costly, reliable, available, and pertinent; but if available and attainable evidence is differentiated along these and other dimensions, then problem-solving can be guided by the ideal of maximum evidence for minimum cost. For example, here is a strategy for focusing attention on available evidence:

CONTEXT: to minimize cost

CONDITIONS: $test_1$ and $test_2$ are pertinent, and
$test_1$ is potentially-confirming, and
$test_2$ is potentially-supporting, and
cost($test_1$) >> cost($test_2$)

ACTIONS: begin
do $test_2$
if supporting then do $test_1$
else do not do $test_1$
end

That is, given cheap, weak evidence and expensive, strong evidence, get the weak evidence first and don't incur the cost of the strong evidence unless the weak evidence lends support. The rule serves to manage the uncertainty associated with the weak evidence – it says seek strong corroboration only if the weak evidence is positive. It also uses features of evidence such as cost and reliability to control the acquisition of evidence; for example, it explains why an angiogram (an expensive, risky, and excrutiating test) is done only after a stress test in Figure 3.1. We distinguish these functions – managing uncertainty and control – only because uncertainty and control have, with a few exceptions noted below, been viewed as different topics. In fact, if control decisions are based on features of evidence, then control and managing uncertainty are the same

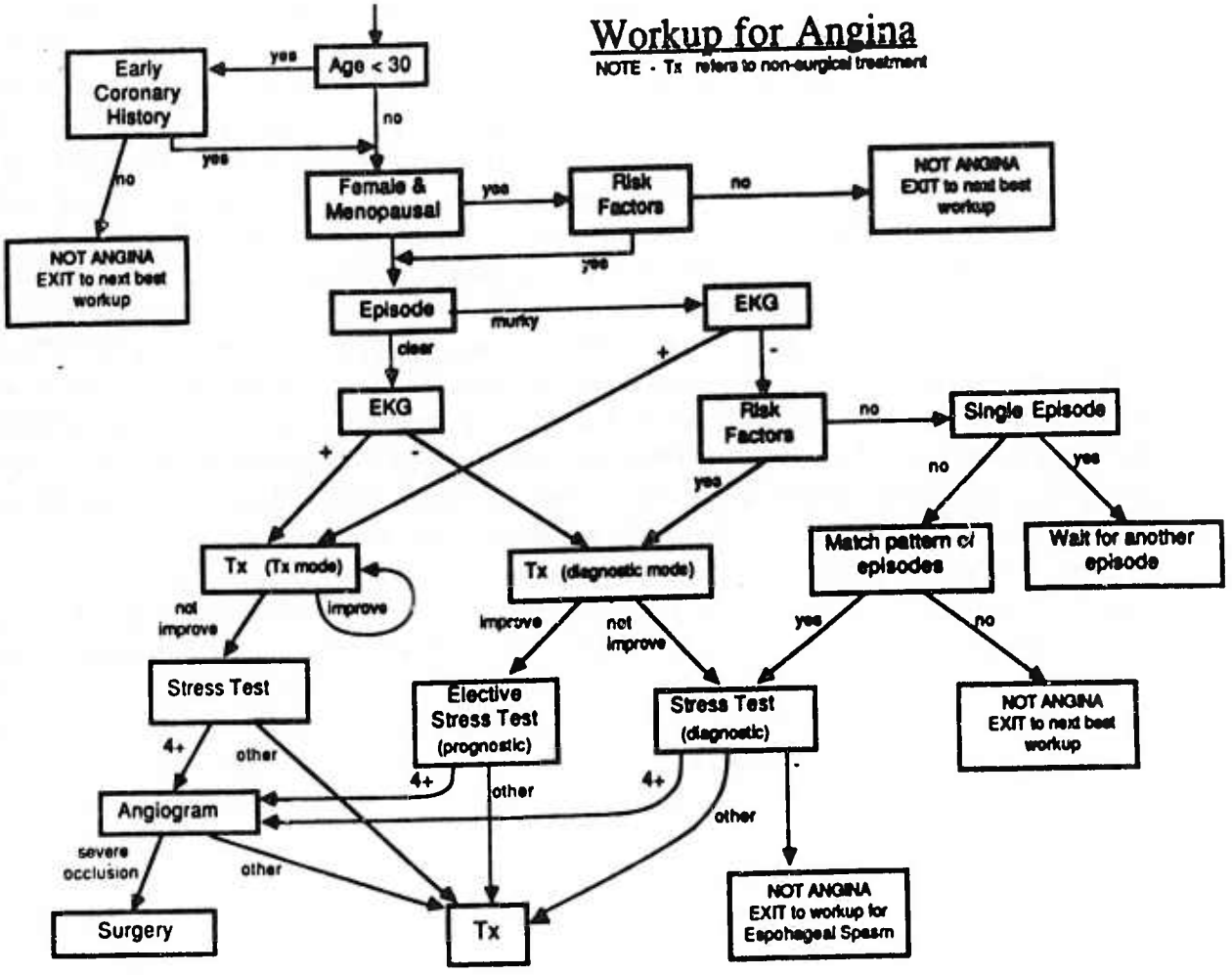

Workup for Angina
NOTE - Tx refers to non-surgical treatment
Age < 30
yes
Early Coronary History
no
yes
no
NOT ANGINA EXIT to next best workup
Female & Menopausal
yes
Risk Factors
no
NOT ANGINA EXIT to next best workup
yes
Episode
murky
EKG
clear
EKG
+
-
+
Risk Factors
no
Single Episode
yes
no
yes
Tx (Tx mode)
Tx (diagnostic mode)
Match pattern c/ episodes
Wait for another episode
not improve
improve
improve
not improve
yes
no
Stress Test
Elective Stress Test (prognostic)
Stress Test (diagnostic)
NOT ANGINA EXIT to next best workup
4+
other
4+
other
4+
other
Angiogram
severe occlusion
other
Surgery
Tx
NOT ANGINA EXIT to workup for Espohageal Spasm

Figure 3.1:

thing. This is the principle that motivates the design of MUM discussed in Section 2.

1.3 Related Work

The close association between control and managing uncertainty has been apparent in the literature on sophisticated control for several years [1] but is largely absent from the AI literature on reasoning under uncertainty. Three important results have emerged from research on control: First, complex and uncertain problems have to be solved *opportunistically* and *asynchronously* – working on subproblems in an order dictated by the availability and quality of evidence (Hayes-Roth and Lesser, 1977). Second, since control tends to be accomplished by *local* decisions about focus of attention, the behavior of complex knowledge systems sometimes lacks global coherence. Coherence can be achieved by *planning* sequences of actions instead of selecting individual actions by local criteria[2]. Third, programs are impossible to understand if the factors that affect control decisions are *implicit*. For example, the focus of attention in Hearsay-II was difficult to follow because it depended on many numerical parameters calculated from data and combined by empirical functions with "tuning" parameters (Hayes-Roth and Lesser, 1977). A better approach is to explicitly state and reason about the implicit factors, called *control parameters* (Wesley, 1983), that the numbers represent (Davis, 1985; Clancey, 1983). If the control parameters are features of evidence and uncertainty, then control strategies can be developed to manage uncertainty.

This last point colors our reading of the AI literature on reasoning under uncertainty. Much of it is concerned with the mathematics of combining evidence, the calculation of *degrees of belief* in hypotheses. (A representative sample includes Shortliffe and Buchanan, 1975; Duda, Hart, and Nilsson, 1976; Zadeh, 1975; Shafer, 1976. See Cohen and Gruber, 1985; and Bonissone, 1985, for literature reviews, including nonnumeric approaches to uncertainty; and Szolovits and Pauker, 1978 for a discussion of uncertainty in medicine.) Degrees of belief *can* serve as control parameters, but it is necessary to maintain a distinction between combining evidence and control. Otherwise, degrees of belief (and the functions that combine them) have to be "tuned" not only to find the most likely answer but also to focus attention in a reasonable way. Inevitably they become ambiguous summaries of implicit control parameters. For example, MYCIN's certainty factors contained probabilistic and salience information, an indirect result of using them to focus attention (Buchanan and Shortliffe, 1985).

Another important reason to maintain the distinction between combining evidence and control is that combining evidence is only a part of the problem of reasoning under uncertainty. Other aspects include formulating decisions, assessing the need for more evidence, planning how to get it, deciding whether it is worth the cost and, if it isn't,

[1] For example, the classic paper by Erman, Hayes-Roth, Lesser, and Reddy (1980) is called "The Hearsay-II speech understanding system: Integrating knowledge to resolve uncertainty."

[2] Personal communication, Victor Lesser.

hedging against residual uncertainty. In MUM we address the problem of combining uncertainty in the context of these other tasks.

2 An Architecture for Managing Uncertainty

Managing uncertainty in MUM requires several kinds of knowledge discussed in this section. Anticipating section 2.3, on control, it may be useful to think of data moving bottom-up through Figure 3.2 as it triggers hypotheses and is requested by MUM's planner.

2.1 Types of Knowledge

Data, Evidence, and Interpretation Functions. Evidence is abstracted from data through interpretation functions. All data about a patient are stored in frames that describe personal history, family history, tests, history of episodes, and other information. Interpretation functions map data to evidence; for example, information that a patient smokes 3 packs of cigarettes a day is abstracted to the evidence *heavy-smoker* by an interpretation function that maps data about smoking habits to one of *(non-smoker light-smoker moderate-smoker heavy-smoker)*. Interpretation functions are often graphs called *belief curves* that relate ranges of a continuous data variable to belief in evidence. Figure 3.3 shows a belief curve relating the duration of chest pain to the evidence *classic-anginal-pain*. Belief curves and other interpretation functions are acquired from an expert. They provide the same functionality as fuzzy predicates (Zadeh, 1975), and generalize Clancey's view of data abstraction as categorical (Clancey, 1983).

Features of Evidence. Evidence may be characterized by its cost, reliability, and roles. The cost of evidence reflects monetary cost as well as discomfort and risk to the patient (later versions of MUM will separate these and other determinants of cost). Reliability refers to several factors, including false-positive and miss rates of tests, and also the belief in evidence derived from belief curves (e.g., is *classic-anginal-pain* at least *supported* by data about the pain duration?) The most important feature of evidence is the *roles* it can play with respect to evaluating hypotheses. MUM recognizes five roles, two of which are symmetric pairs:

Potentially-confirming and potentially-disconfirming. If evidence plays a potentially-confirming role with respect to a hypothesis, then acquiring it *might* confirm the hypothesis, though not all potentially-confirming evidence will, in actuality, confirm. For example, an EKG confirms the hypothesis of angina only if "positive" (i.e., shows ischemic changes.) Once confirmed (or disconfirmed), a hypothesis

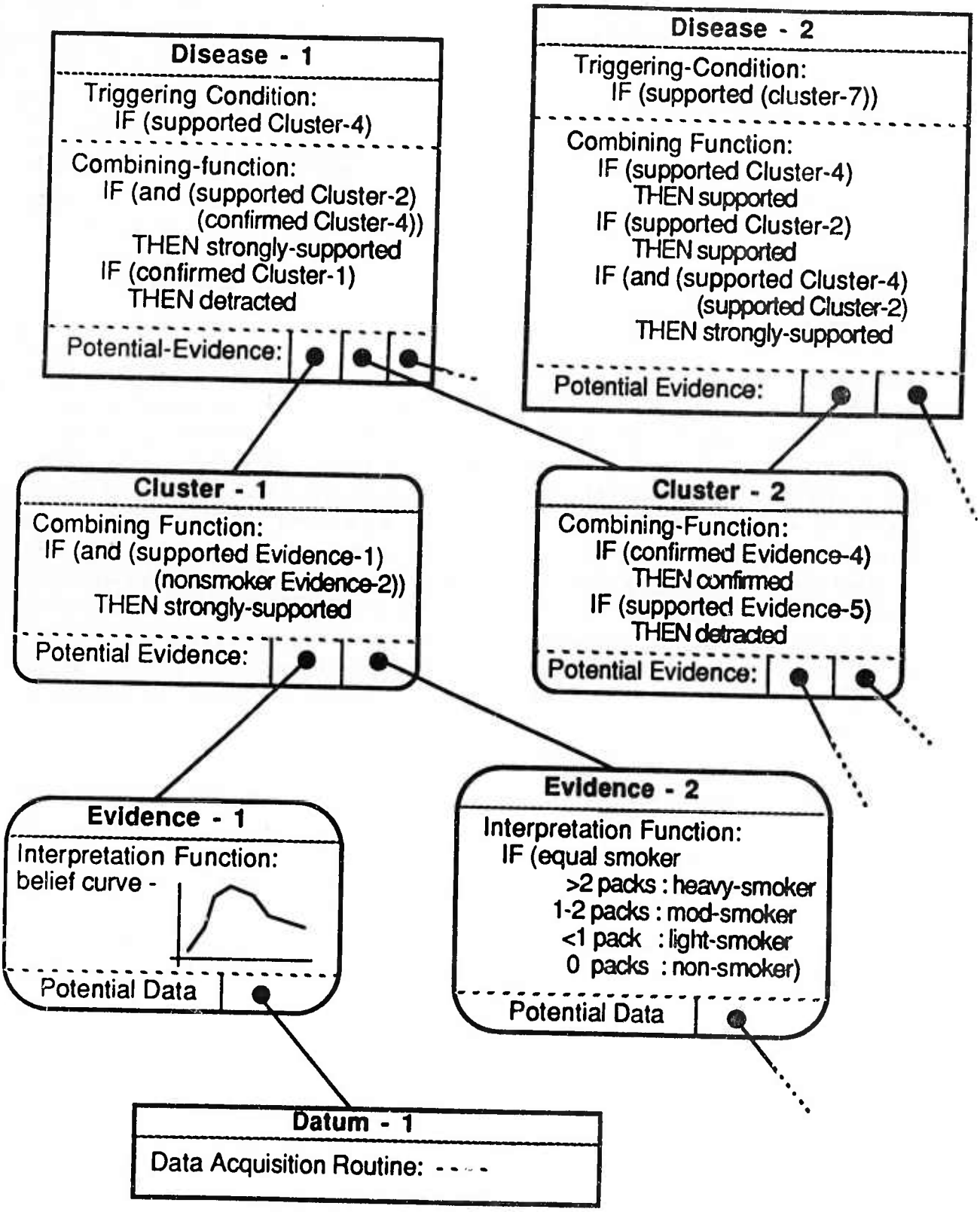


FIGURE 3.2: Knowledge Structures in MUM

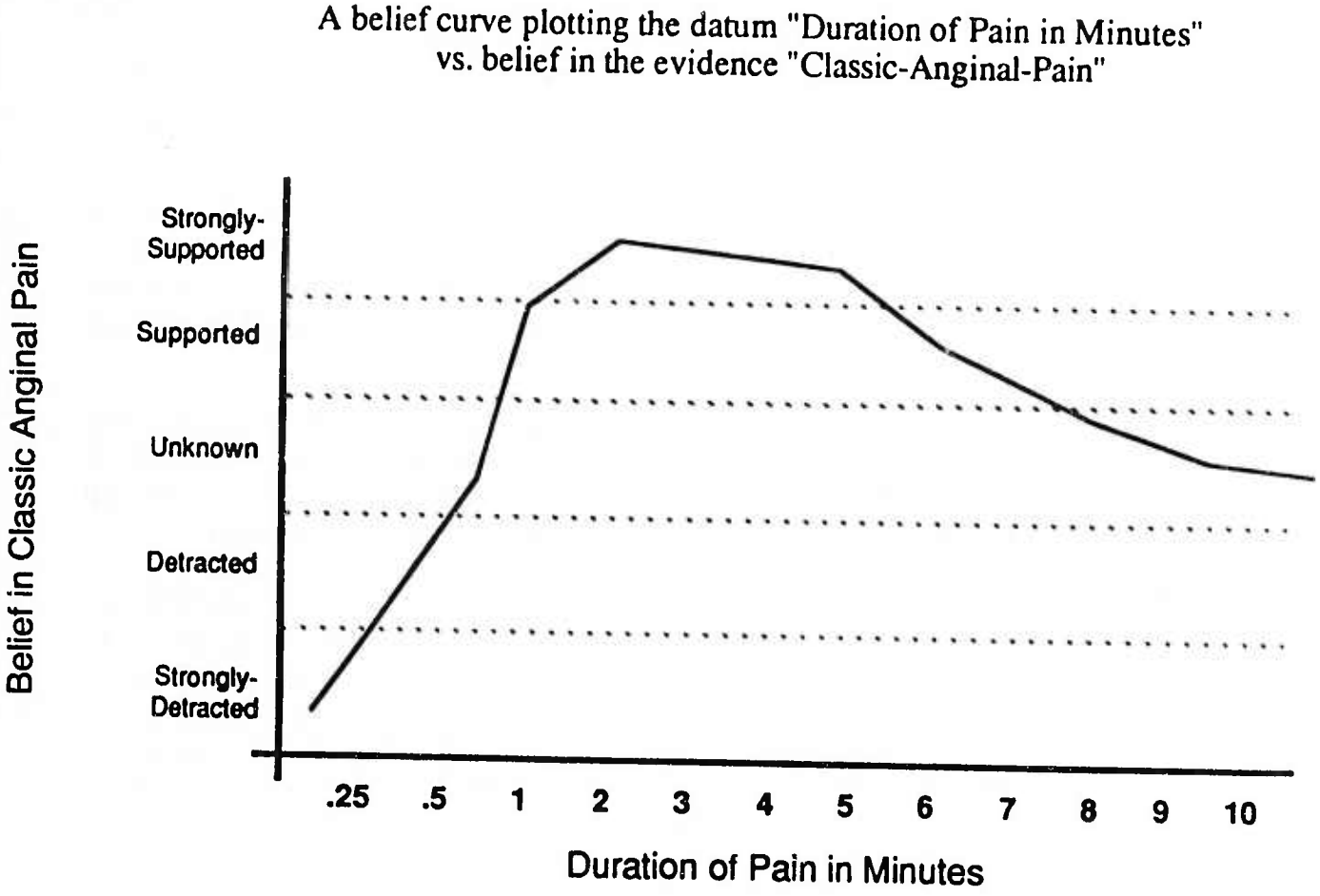

A belief curve plotting the datum "Duration of Pain in Minutes"
vs. belief in the evidence "Classic-Anginal-Pain"
Belief in Classic Anginal Pain
Strongly-Supported
Supported
Unknown
Detracted
Strongly-Detracted
.25
.5
1
2
3
4
5
6
7
8
9
10
Duration of Pain in Minutes

Figure 3.3:

requires no further evidence, though a diagnostician may continue working to disconfirm other hypotheses, especially if they are dangerous.

Potentially-supporting and potentially-detracting. Like *potentially-confirming* and *potentially-disconfirming*, but not conclusive. However, combinations of supporting or detracting evidence may be confirming and disconfirming, respectively (see "Combining Functions," below). The combination referred to as cluster-2 (Figure 3.2) is *potentially-supporting* with respect to disease-2; cluster-1 is *potentially-detracting* with respect to disease-1.

Trigger. Evidence plays the *triggering* role with respect to a hypothesis if its presence focuses attention on the hypothesis, or "brings the hypothesis to mind," or, in MUM, adds the hypothesis to a list of potential diagnoses. Cluster-4, if it is *supported*, triggers disease-1 (Figure 3.2). This role of evidence is found in virtually all medical expert systems.

Modifying. Some evidence primarily alters the way diagnosis proceeds. For example, risk factors for coronary artery disease (e.g., hypertension, elevated cholesterol) play a *modifying* role with respect to the hypothesis of angina since diagnosis will proceed aggressively if they are present and less aggressively otherwise.

Note that evidence can play multiple roles with respect to any hypothesis; for example, risk factors are both *potentially-supporting* and *modifying* with respect to angina; and most *triggers* are individually or in combination with other evidence at least *potentially-supporting* (e.g., note the roles cluster-4 plays with respect to disease-1 in Figure 3.2). Also, one piece of evidence can play different roles with respect to several hypotheses (illustrated by the roles cluster-2 plays with respect to disease-1 and disease-2 in Figure 3.2). Finally, note that some evidence potentially plays two symmetric roles, while some is "asymmetric"; for example, a stress test will either support coronary artery disease or detract from it, while an EKG supports angina if it is positive and is useless otherwise. That is, EKG plays a *potentially-supporting* role only.

Clusters. Physicians often see collections of evidence that play particular roles in diagnosis; for example, shortness of breath that comes on suddenly but is unrelated to exercise or other inciting factors *triggers* the diagnosis of pulmonary embolism. Just as evidence has roles with respect to clusters, so clusters have roles with respect to diseases, and these roles need not be supporting; for example, the cluster *(patient-age < 30 and no-family-history-of-coronary-events)* plays a *potentially-disconfirming* role with respect to all coronary diagnoses of chest pain. Instead of saying that the available evidence is a poor match to coronary diagnoses, we can say the evidence is a good match to a cluster that potentially detracts from or disconfirms coronary diagnoses.

Combining Functions. Every cluster includes a function, specified by the expert, that combines the available evidence for the cluster and returns a value for the cluster given evidence. The values returned by combining functions are just "realizations" of potential roles of evidence. For example, the value returned by the combining function of a cluster supported by *potentially-confirming* evidence could be *confirmed*. The value for a cluster with several pieces of *potentially-detracting* evidence might be *disconfirmed*, or perhaps *detracted*. Combining functions are further discussed below.

Diseases. A disease is technically a cluster. Diseases reside at the top of a hierarchy of clusters (as shown in Figure 3.2), each of which has its own combining function and specifications of the roles played by the clusters below it.

Strategic Knowledge. We characterize strategic knowledge as heuristics for deciding which triggered disease hypotheses to focus on, and how to go about selecting actions to gather evidence pertinent to these hypotheses. These heuristics have the same contingent nature as Davis' meta-rules (Davis, 1985) and control rules in Neomycin (Clancey, 1985). We represent strategies as rules that include:

- *conditions* for selection of the strategy;
- a *focus policy* that guides the choice of a subset of the triggered disease hypotheses to focus on;
- *planning criteria* that guide the selection of actions to gather evidence and treat diseases currently in the focus.

Examples of focus policies are *plausibility* (choose hypotheses based on their degree of support); *criticality* (focus on hypotheses that, if true, would require immediate action); and *differential* (focus on hypotheses that offer alternate explanations for the symptoms). Examples of planning criteria are *cost* (prefer evidence that is easy to obtain, and inexpensive on some cost metric); *roles* (prefer potentially-confirming over potentially-supporting); and *diagnosticity*, meaning that a given result has the potential to increase the belief in one hypothesis and decrease belief in the other, as indicated by belief curves.

2.2 Combining Evidence and Propagating Belief

In MUM, evidence is combined by local functions, as shown in Figure 3.2. Typically, knowledge systems require three functions to combine evidence and propagate belief. These are illustrated in the context of two inference rules:

R1: (A AND B) → C
R2: (D AND E) → C

One function calculates the degree of belief (dob) in a conjunction from degrees of belief in the conjuncts:

$$\mathbf{dob}(\text{AND A B}) = F_1(\mathbf{dob}(A)\mathbf{dob}(B))$$

The second function calculates the degree of belief in a conclusion from

a) the degree of belief in its premise (computed by F_1)

b) the "conditional" degree of belief in the conclusion given the premise; often called the degree of belief in the inference rule:

$$\mathbf{dob}(\mathrm{C}_{R1}) = F_2(\mathbf{dob}(\text{AND A B}), \mathbf{dob}(\mathrm{C}|(\text{AND A B})))$$

The third increases the degree of belief in a conclusion when it is derived by independent inferences:

$$\mathbf{dob}(C_{R1\&R2}) = F_3(\mathbf{dob}(C_{R1}), \mathbf{dob}(C_{R2}))$$

In MUM, these three kinds of combining are maintained, but with two important differences. First, there are no *global* functions corresponding to F_1, F_2, and F_3; all combining is done by functions local to clusters. Second, instead of the usual numeric degrees of belief, MUM has seven levels of belief: *disconfirmed, strongly-detracted, detracted, unknown, supported, strongly-supported, confirmed.* These are just "realizations" of the roles of evidence described earlier.

Combining evidence and propagating belief in MUM is illustrated in Figure 3.2. Each cluster, including diseases, has its own local combining function, specified by an expert. For example, cluster-1 is *strongly-supported* if the data *support* evidence-1 and if the data on a patient's smoking habits support evidence that he or she is a nonsmoker. This is a conjunction of evidence of the kind calculated by F_1, above. Another example is found in the combining function for disease-1. If cluster-2 and cluster-4 are both *confirmed*, then disease-1 is *strongly-supported*. This illustrates the kind of combining for which F_2, above, is required: even when the evidence for a disease is itself certain, the conditional belief in the disease given the evidence may not be certain. Disease-2 also contains a conjunctive rule, but the entire combining function illustrates the corroborative situation for which F_3 is needed. In this case, cluster-4 and cluster-2 *individually* play *potentially-supporting* roles, and taken together increase the level of belief in disease-2 to *strongly-supporting*.

Local combining functions have many advantages. Foremost is the ease with which an expert can specify *precisely* how the level of belief in a cluster depends on the levels of belief in the evidence for that cluster. Control of combining evidence is not relinquished to an algorithm, but is acquired from the expert as part of his or her expertise. Since

local combining functions are specific to clusters, they can be changed independently. And since the values passed between them in MUM are few, it is easy to trace back the derivation of a level of belief and pinpoint a faulty local combining function. The prospect of having to acquire many functions seems daunting, but we have found it easy and intuitive, and much easier to explain than a global numeric method.

2.3 Control of Diagnosis in MUM

MUM's basic control strategy involves three components:

User Interface: uses data description frames in the knowledge base to ask questions and create patient data frames for the results;

Matcher: uses the interpretation and combining functions to record the effect incoming data have on the belief states for clusters and disease frames, and triggers new hypotheses as appropriate;

Planner: uses strategic control rules to guide the selection of a focus set and the planning process.

Basic Control. The planner follows a basic control loop within which it interprets strategic control rules. It is implemented in a blackboard system with knowledge sources specified in the same syntax as strategic control rules. This facilitates experimental modifications. The design of the blackboard system was influenced by Hayes-Roth (1985), and shares the emphasis on explicit solution to the control problem. We first describe the basic control loop, then strategies and their selection.

The basic control loop is initiated with the choice of a *strategic phase*. All strategic phases but one include a *focus policy* that directs MUM's attention to a subset of candidate hypotheses. This is followed by the selection of short-term plans to gather evidence and select treatment pertinent to these hypotheses (the rule in Section 1.2 represents such a plan). Since the effort of developing lengthy plans may well be wasted in a domain permeated with uncertainty, we constrain plans to single actions or sequences of two actions where the applicability of the second depends on the outcome of the first. Several short-range plans may be generated and executed.

Carrying out plans typically consists of invoking the user interface to request some information, updating the status of the diseases with the matcher, and conditional continuation of the plan. When no short-term plans remain, the system iterates the basic control loop to determine if a new strategic phase is appropriate, updates the focus, and generates new short-term plans.

Strategic Control. We represent MUM's overall strategy as an ordered set of rule-like strategic phases, shown in Figure 3.4. Each phase has conditions that activate it.

Once activated, a phase controls MUM's focus of attention and the choice of actions pertaining to the hypotheses in this focus.

The phase **Get General Picture** is invoked when the system is started, and may also be used if all previously considered hypotheses are ruled out. It has no focus policy because no hypotheses are active when it is invoked. It directs the planner to ask for evidence that plays the **potential-trigger** role for one or more hypotheses, pursuing the lowest-cost evidence first. The cluster **initial-consultation** (consisting of age, sex, and primary complaint) meets the criteria of potentially triggering many hypotheses and costing little. The initial consultation usually triggers some hypotheses, which result in a new strategic phase being selected. If no hypotheses were triggered, the planner asks for potential-triggers of higher cost.

The **Initial Assessment for Triggered Hypotheses** phase is invoked when new hypotheses are triggered. Since the conditions of the other strategic phases depend somewhat on the level of belief in candidate hypotheses, this phase gathers preliminary evidence for the hypotheses. The focus is on the triggered hypotheses, so only evidence playing some role relative to these hypotheses is considered by the planner. This phase directs the planner to gather low-cost evidence for the hypotheses. For example, MUM asks about aspects of the patient's episode (the event which is the primary complaint) which bear on the triggered hypothesis, and about risk factors.

As soon as the easy questions for triggered hypotheses have been asked, MUM decides between the next two phases on based its belief in the hypotheses and whether any of the hypotheses are *critical*, that is, require immediate treatment if supported. Critical hypotheses are dealt with first.

The **Deal With Critical Hypotheses** phase places all candidate critical hypotheses in MUM's focus. The short range planner is then directed to attempt to rule out these hypotheses. It begins with potentially-disconfirming or potentially-detracting evidence. If it fails to find any, then it looks for potentially-supporting evidence. It will not seek evidence that plays a lesser potential role than evidence it already has. For example, it will not seek potentially-supporting evidence for a hypothesis that is already strongly supported, but rather focuses on potentially-confirming evidence. The planner will focus on low-cost evidence first, but it is not prohibited from pursuing high-cost evidence as it was in the previous phase.

If the focus of attention is not captured by critical hypotheses, it is dictated by plausibility. The strategic phase **Discriminate Strongest Hypotheses** discriminates competing alternatives with as little cost to the patient as possible. As before, the potential roles of evidence are used to decide whether it is worth acquiring.

Currently MUM stops work when a hypothesis is confirmed and no critical hypotheses remain in its focus. We are implementing the next strategic phases, prognosis and treatment. Both provide evidence of diagnostic significance; for example, MUM may begin treatment for angina if it is strongly supported, rather than incurr the cost of absolute confirmation. If the treatment relieves the symptoms, then it is additional

Strategic Phase:	Get General Picture.
Conditions:	No candidate hypotheses.
Focus Policy:	None.
Planning Criteria:	Evidence must play trigger role; prefer low cost on all cost metrics.
Strategic Phase:	Initial Assessment for Triggered Hypotheses.
Conditions:	One or more hypotheses are triggered.
Focus Policy:	Focus on triggered hypotheses.
Planning Criteria:	Must be low on all cost metrics; prefer stronger roles.
Strategic Phase:	Deal With Critical Possibilities
Conditions:	There are critical hypotheses which have not been confirmed, disconfirmed or strongly detracted, and if they are detracted, no other hypothesis is confirmed.
Focus-Policy:	Criticality.
Planning Criteria:	Rule Out if possible, else gather support. Utility of evidence. Low cost first; as needed let discomfort and monetary cost increase.
Strategic Phase:	Discriminate Strongest Hypotheses
Conditions:	More than one hypothesis is supported.
Focus-Policy:	Plausibility.
Planning Criteria:	Diagnosticity, Low cost first. Utility of evidence. Substitute high cost confirmation for one hypothesis with lower cost disconfirmation for the other.

Figure 3.4: Four Strategic Phases in MUM's Diagnosis

evidence for the diagnosis. If not, it is evidence that detracts from the diagnosis and may support others. Since treatment provides evidence, we represent treatments as clusters, exactly the same way as we represent tests such as angiography.

The emphasis in MUM is on asking the right questions in the right order without superfluous questions. MUM's control knowledge is not yet sophisticated enough to satisfy all these criteria. It asks questions in a reasonable order, but it sometimes focuses on the wrong disease. Since MUM is a nascent system, this does not yet concern us. We believe the system is successful in providing a framework for exploring management of uncertainty by sophisticated control, that is, by making control decisions based on the roles, costs and other characteristics of evidence; and the criticality of diseases; and the credibility of diagnoses.

3 Conclusions

MUM manages uncertainty by reasoning about evidence and its current state of belief in hypotheses. Its goal is to generate appropriate workups for chest and abdominal pain, that is, to ask the right questions in the right order without unnecessary questions. To the extent it succeeds, it demonstrates its ability to manage uncertainty, to select the appropriate action given uncertainty. We have said this is a control task. Indeed, much of MUM's architecture is devoted to explicit, evidence-based control.

Much work remains to be done. Currently, MUM resembles a programming environment more than a medical expert system. We will be devoting ourselves to building up its clusters and control rules.

Although MUM was designed for medical problems and is discussed in that context, we believe the approach to uncertainty and control it engenders is general to classification problem solvers, as well as to other systems responsible for the management of uncertainty. An empty version of MUM called MU has been developed and is discussed in the next section.

4 The MU Architecture

MU is a development environment for knowledge systems that reason with incomplete knowledge. It has evolved from a program called MUM that planned diagnostic sequences of questions, tests, and treatments for chest and abdominal pain (Cohen et al., 1987). This task is called prospective diagnosis, because it emphasizes the selection of actions based on their potential outcomes and the current state of the patient. Prospective diagnosis is uncertain because the precise outcomes of actions cannot be predicted, in part because knowledge of the state of the patient is incomplete. Yet we have found that physicians have rich strategic knowledge with which they plan diagnoses in spite of their their uncertainty. MU does not provide a knowledge engineer

with any particular strategies, but rather provides an environment in which it is easy to acquire, represent, and experiment with a wide variety of strategies for prospective diagnosis and other *prospective reasoning* tasks.

Three goals underlie our research and motivate the MU system. First, MU is intended to provide knowledge-engineering tools to help acquire expert problem-solving strategies. MU allows us to define explicit *control features*, which are the terms an expert uses to discuss strategies. Control features in medical diagnosis include degrees of belief in disease hypotheses, monetary costs of evidence, the consequences of incorrect conclusions, and "intangibles" such as anxiety and discomfort. Some, like degrees of belief, have values that change dynamically during problem solving. MU helps the knowledge engineer define the functions that compute these dynamic values and keeps the values accessible during problem solving. For example, with MU we can easily define a control feature called *criticality* in terms of two others, say *dangerousness* and *degree of belief*, and acquire a function for dynamically assessing the criticality of a hypothesis as its degree of belief changes.

Second, we want to show that strategies enable a prospective reasoning system to produce solutions that are *efficient* in the sense of minimizing the costs of attaining given levels of certainty. MU has no "built in" problem solving strategies, but we have been able to acquire and implement efficient, expert strategies in MU because we can define explicit control features that represent the various costs of actions, as well as the levels of certainty in the evidence produced by actions.

Third, we want to implement in MU a *task-level architecture* for prospective reasoning (Gruber and Cohen, 1987), an environment for building systems that plan efficient sequences of actions, despite uncertainty about their outcomes. After working in the domains of medicine and plant pathology, we now think that many control features pertain to diagnostic tasks in general. Moreover, diagnosticians in many fields seem to use similar strategies to solve problems efficiently. This view is influenced by the recent trend in AI toward defining *generic tasks* (Chandrasakeran, 1986) such as classification (Clancey, 1985) and the architectures that support their implementation. MU shares the orientation toward explicit control efforts such as BB* (Hayes-Roth, 1985, Hayes-Roth et al., 1986) and Heracles (Clancey, 1986) but emphasizes control features that are appropriate for prospective reasoning.

In sum, MU is a tool for representing and providing access to the knowledge that underlies efficient prospective reasoning. This report begins with an analysis of prospective reasoning, then describes the MU environment first as a program, emphasizing its structure and function, then from the perspective of the knowledge engineer who uses it. As an illustration, we describe how MUM was reimplemented in MU. We conclude with a summary of current work.

5 Prospective Reasoning

Prospective reasoning is reasoning about the question "What shall I do next," given that

1. knowledge about the current state of the world is incomplete,
2. the outcomes of actions are uncertain,
3. there are tradeoffs between the costs of actions with respect to the problem solver's goals and the utility of the evidence they provide,
4. states of knowledge that result from actions can influence the utility of other actions.

An example from medical diagnosis illustrates these characteristics:

> A middle-aged man reports episodes of chest pain that could be either angina or esophageal spasm; the physician orders an EKG, but it provides no evidence about either hypothesis; then he prescribes a trial prescription of vasodilators; the patient has no further episodes of pain, so the physician keeps him on long-acting vasodilators and eventually suggests a modified stress test to gauge the patient's exercise tolerance.

The first and second characteristics of prospective reasoning are clearly seen in this case: Knowledge about the state of the patient is incomplete throughout diagnosis, and the outcomes of actions (the EKG, trial therapy, stress test) are uncertain until they are performed and are sometimes ambiguous afterwards. Less obvious is the third characteristic, the tradeoffs inherent in each action. Statistically, an EKG is not likely to provide useful evidence, but if it does, the evidence will be completely diagnostic. The EKG is given because its minimal costs (e.g., time, money, risk, and anxiety) are offset by the possibility of obtaining diagnostic evidence[3]. Similarly, trial therapy satisfies many goals; it protects the patient, costs little, has few side-effects and, if successful, is good evidence for the angina hypothesis.

The fourth characteristic of prospective reasoning is that states of knowledge that result from actions can affect the utility other actions. This is because the costs and benefits of actions are judged in the context of what is already known about the patient. For example, trial therapy is worthwhile if the EKG does not produce diagnostic evidence, but is redundant otherwise. The outcome of an EKG thus affects the utility of trial therapy. This implies a dependency between the actions, and suggests a strategy: do the EKG first because, if it is positive, then trial therapy will be unnecessary.

[3]This example oversimplifies the reasons for giving an EKG, but not the cost/benefit analysis that underlies the decision.

Dependencies between actions help the prospective reasoner to *order* actions. We call this planning, though it is not planning in the usual AI sense of the word (Sacerdoti, 1979) (Cohen and Feigenbaum, 1982). The differences are due to the first and second characteristics of prospective reasoning: the state of the world and the effects of actions are both uncertain. The prospective planner must "feel its way" by estimating the likely outcomes of one or more actions, executing them, then checking whether the actual state of the world is as expected. Plans in prospective reasoning tend to be short. In contrast, uncertainty is excised from most AI planners by assuming that the initial state of the world and the effects of all actions are completely known (e.g., the STRIPS assumption, (Fikes, Hart, and Nilsson, 1972). AI planners can proceed by "dead-reckoning," because it follows from these assumptions that *every* state of the world is completely known. All further discussions of planning in this report refer to the "feel your way" variety, not to "dead reckoning."

Prospective diagnosis requires a planner to select actions based on their costs and utility given the current state of knowledge about the patient. We have described prospective reasoning as planning because the evidence from one action may affect the utility of another. Alternatively, prospective reasoning can be viewed as a series of decisions about actions, each conditioned on the current state of knowledge about the patient. We considered decision analysis (Raiffa, 1970, Howard, 1966) as a mechanism for selecting actions in prospective reasoning, but rejected it for two reasons. First, collapsing control features such as monetary expense, time, and criticality into a single measure of utility negates our goals of explicit control and providing a task-level architecture for prospective reasoning (Cohen, 1985, Gruber and Cohen, 1987). Second, decision analysis requires too many numbers — a *complete*, combinatorial model of each decision. The expected utility of each potential action can only be calculated from the joint probability distribution of the possible outcomes of the previous actions. But although we do not implement prospective reasoning with decision analysis, MU is designed to provide qualitative versions of several decision-analytic concepts, including the utility of evidence and sensitivity analysis.

6 The MU Environment – An Overview

A coarse view of MU's structure reveals these components:

- a frame-based representation language,
- tools for building inference networks,
- an interface for defining control features and the functions that maintain their values,

An Inference Net in MU

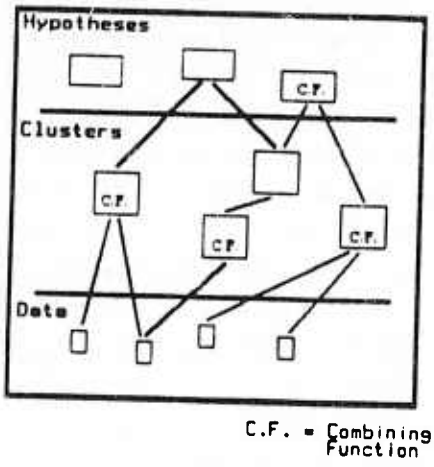


Figure 3.5: Organization of Knowledge Within Mu

- a language for asking questions about the state of a problem and how to change its state.
- a user interface for acquiring data during problem-solving,

With these tools, a knowledge engineer can build a knowledge system with a planner for prospective reasoning. MU does not "come with" any particular planners, but it provides tools for building planners and incorporating expert problem-solving strategies within them.

Among MU's tools is an editor for encoding domain inferences, such as *if EKG shows ischemic changes then angina is confirmed*, in an inference network. MU does not dictate what the nodes in the inference network should represent, except in the weak sense that nodes "lower" in the network — relative to the direction of inference — provide evidence for those "higher" up. However, the nodes in the network are usually differentiated; for example, in Figure 3.5 some nodes represent raw data, others represent combinations of data (called *clusters*), and a third class represents hypotheses. In the medical domain, data nodes represent individual questions, tests, or treatments. Clusters combine several data; for example, the *risk-factors-for-angina* cluster combines the patient's blood pressure, family history, past medical history, gender, and so on. Hypothesis nodes represent diseases such as *angina*.

Since MU does not provide a planner, the knowledge engineer is required to build one. The planner should answer two questions:

- Which node(s) in the network should be in the focus set, and which of these should be the immediate focus of attention?

- Which actions are applicable, given the focus set, and which of these should be taken?

For example, in the medical domain the focus set might include all disease hypotheses that have some support, and the immediate focus of attention might be the most dangerous one. The potential actions might be the leaf nodes of the tree rooted at the focus of attention (Figure 3.5), and the selected action might be the cheapest of the potential actions.

MU provides an interface to help the knowledge engineer define control features such as the degree of belief in hypotheses, the dangerousness of diseases, and the costs of diagnostic actions. It also provides a language with which a planner can query the values of features and ask about actions that would change those values. Planners can ask, for example, "What is the current level of belief in angina?" or "Tell me all the inexpensive ways to increase the level of belief in angina," or even the hypothetical question, "Would the level of belief in angina change if blood pressure was high?"

The relationship between these functions of MU and the functions of a planner are shown in Figure 3.6. Using MU, a knowledge engineer can: define a control feature such as criticality in terms of other features such as dangerousness and degree of belief; specify a combining function for calculating dynamically the value of criticality from these other features during problem solving; associate criticality and its combining function with a class of nodes, such as diseases, and have each member of the class inherit the definitions; and write a planner that encodes an expert strategy for dealing with critical or potentially-critical diseases. MU facilitates the development of planners, and makes their behavior explicit and efficient, but the *design* of planners, and the acquisition of strategies and the control features on which they depend, is the job of the knowledge engineer.

7 The MU Environment – Features and Combining Functions

Knowledge representation in MU centers around features. Features and their values are the information with which planning decisions are made. Each node in a MU inference network can have several features; for example, the node that represents trial therapy for angina includes features for monetary cost and risk to the patient. Features are defined in the normal course of knowledge engineering to support expert strategies for prospective reasoning. We have identified four classes of features, differentiated by their value types, how they are calculated, and the operations that MU can perform on them:

Static The value of a static feature is specified by the expert and does not change at run time. *Monetary cost* is a typical static feature, as the cost of an action does

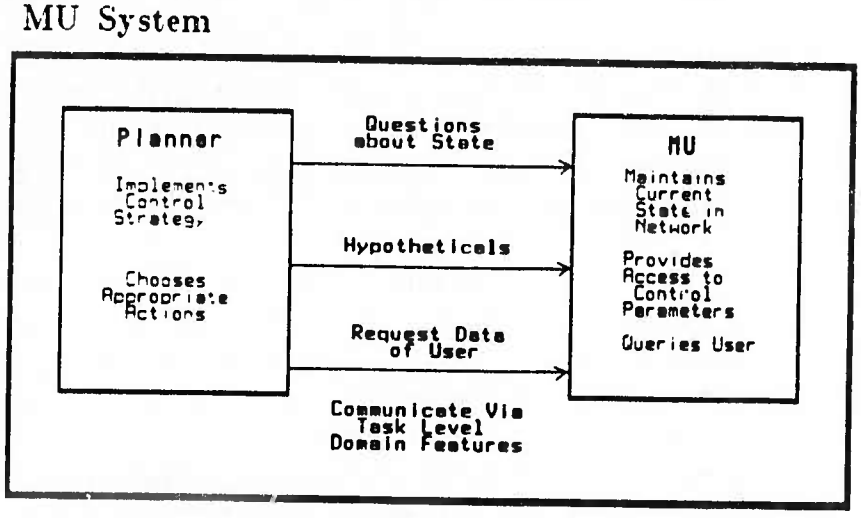


Figure 3.6: Mu System Schematic

not change during a session.

Datum The value of a datum feature is acquired at run time by asking the user questions. Data are often the results of actions; for example *EKG shows ischemic changes* is a potential result of performing an EKG.

Dynamic The value of a dynamic feature is computed from the values of other feature values in the network. The value of each dynamic feature is calculated by a combining function, acquired through knowledge engineering. A dynamic feature of every hypothesis is its *degree of belief* — a function of the degrees of belief of its evidence.

Focus The value of a focus feature is a set of nodes whose features satisfy a user-defined predicate. Focus features are a subclass of dynamic features. In medicine, the *differential* focus feature can be defined as the list of all triggered hypotheses that are not confirmed or disconfirmed.

Feature values can belong to several data types, including integers, sets, normal (one of an unordered set of possible values), ordinal (one of an ordered set of possible values), boolean, and relational (e.g., isa).

Four operations are defined for features: one can *set* a feature value (e.g., assert that the monetary cost of a test is high) *get* a feature value (e.g., ask for the cost of a test), ask *how to change* a feature value, and ask *what are the effects* of changing a feature value. Planners need answers to these kinds of questions to help them select actions (see Section 5 for further examples.)

	Data Types				Questions			
Feature	Number	Set	Ordinal	Normal	Get	Set	How To	Effect Of
static	X	X	X	X	X			
datum	X	X	X	X	X	X		X
dynamic	X		X	X	X		X	
focus		X			X			

Figure 3.7: Capabilites By Feature Type

All combinations of feature type, value type, and operations are not possible. Figure 3.7 summarizes the legal combinations.

MU provides an interface for defining features. A full definition includes the feature type, value type, its range of values, and the domain of its combining functions. For instance, the dynamic feature *level of support* is defined to have seven values on an ordinal scale: disconfirmed, strongly-detracted, detracted, unknown, supported, strongly-supported and confirmed. Figure 3.8 shows the definition of level of support.

Level-Of-Support

Feature-type: Dynamic

Value-Type: Ordinal

Value-restriction: (disconfirmed strongly-detracted detracted unknown supported strongly-supported confirmed)

Combination-function-slot: *local to each hypothesis*

Value: *the current level of support of the hypothesis*

Figure 3.8: Definition of Level-Of-Support

Instances of this feature (and others) are associated with individual hypotheses, each of which may have its own, local function for calculating level of support, and its own, dynamic value for the feature[4]. For example, Figure 3.9 shows part of the frame for the angina hypothesis, encompassing an instance of the level of support feature, and showing a fragment of the function for calculating its value for angina.

Combining functions calculate values for dynamic features such as level of belief, criticality, elapsed time, and so on. They serve two important functions: First, they keep the state of MU's inference network up-to-date; for example, when the result of

[4]Not all feature values are calculated locally, but, for reasons discussed in (Cohen, et al., 1987) and (Cohen et al., MUM) levels of belief are.

Angina

Feature-list: (level-of-support severity)

Current-level-of-support: strongly-supported

Combination-function:

IF value of ekg is ischemic-changes
THEN angina is confirmed
ELSEIF episode-incited-by contains exertion AND
    risk-factors-for-angina are supported
THEN angina is strongly-supported . . .

Figure 3.9: Part of the Angina Frame With Local Combining Function

an EKG becomes available, the combining function for the angina node updates the value of its level of support feature accordingly.

Second, and perhaps more important from the standpoint of a planner, combining functions provide a *prospective* view of the effects of actions; for example, the combining function for angina can be interpreted prospectively to say that EKG can potentially confirm angina. The same point holds for the combining functions for other features: MU can prospectively assess the potential effects of actions on all dynamic features. A planner can ask MU, "If EKG is negative, what changes?" and get back a list of all the features of all data, clusters, and hypotheses that are in some way affected by the value of EKG. The effects of actions are assessed in the context of MU's current state of knowledge (i.e., the state of its network). For example, if an EKG has been given and its results were negative, then MU knows that the answer to the previous question is that nothing changes.

The syntax of combining functions is relatively unimportant provided they are declarative, so MU's question-answering interface can read them, and experts can easily specify and modify them. Currently, combining functions look like rules, but we are experimenting with tabular and graphic forms (Cohen, Shafer and Shenoy, 1987).

The two major classes of combining functions are *local* and *global*. A local function for a node such as angina refers only to the nodes in the inference network that are directly connected to angina. In contrast, global functions survey the state of MU's entire inference network. Functions for focus features take a global perspective because the value of a focus feature is the subset of nodes in the network whose features satisfy some predicate. For example, Figure 3.10 illustrates the combining function for the *differential* focus feature. Any node that represents a disease hypothesis, and is triggered, but is neither confirmed nor disconfirmed is a member of the differential.

Differential

feature-list: (focus-feature)

current-focus: (angina prinz-metal ulcer)

combining-function:

Set-of $node$ member-of disease Such-that
$node$ is triggered AND
level-of-support of $node$ is not confirmed AND
level-of-support of $node$ is not disconfirmed

Figure 3.10: Part of the Global Focus-Feature **Differential**

The knowledge engineer can define many focus features, each corresponding to a class of nodes that a planner may want to monitor. Besides the differential, a planner might maintain the set of critical hypotheses (e.g., all dangerous hypotheses that have moderate support or better), or the set of hypotheses that have relatively high prior probability, or the set of all supported clusters that potentially confirm a particular hypothesis. MU supports set intersection, union, and sorting on the sets of nodes maintained by focus features. A planner's current focus of attention is represented in terms of the results of these operations.

3 MU from the Knowledge Engineer's Perspective

MU is a development environment for prospective reasoning systems. We began our research on prospective reasoning when we were building a system, MUM, for prospective diagnosis (Cohen et al, 1987), and realized that we lacked the knowledge engineering tools to acquire and modify diagnostic strategies. An example will illustrate the knowledge engineering issues in building MU:

MUM had several *strategic phases*, each of which specified how to assess a focus of attention and select an action. One phase, called *initial assessment*, directed MUM to focus on triggered hypotheses one by one and take inexpensive actions that potentially support each. This covered a wide range of situations, and maintained the efficiency of diagnoses by focusing on low-cost evidence, but it made little sense for very dangerous disease hypotheses. For these, diagnosticity — not cost — is the most important criterion for selecting actions. Once the expert explained this, we should have immediately added a new strategic phase, run the system, and iterated if its performance was incorrect. Unfortunately, control features such as criticality and diagnosticity did not have declarative representations in MUM, were implemented in lisp, and could not easily be composed from other control features. Operations such as sorting a list of critical hy-

potheses by their level of support were also implemented in lisp. Each strategic phase required a day or two to write and debug. From the standpoint of the expert, it was an unacceptable delay.

The MUM project showed us that MU should facilitate acquisition of control features, maintain their values efficiently, and support a broad range of questions about the state of the inference network. MU allows a planner to ask 6 classes of questions:

Questions about **state** are concerned with the current values of features. For example:

Q1: "What is the current level of support for angina?"

Q2: "Is an ulcer dangerous?"

Q3: "What is the cost of performing an angiogram?"

Another class of questions is asked to find out how to achieve a goal. Examples of questions about goals are:

Q4: "Given what I know now, which tests might confirm angina?"

Q5: "What are all of the tests that might have some bearing on heart disease?"

These questions help a planner identify relevant actions and select among them. Those that pertain to levels of belief are answered by refering to the appropriate combining functions and current levels of belief. For example, the answer to the question about angina is "EKG," if an EKG has not already been performed (Figure 3.9).

Questions about the **effects** of actions allow a planner to understand the ramifications of an action. For example,

Q6: "Which disease hypotheses are affected by performing an EKG?

Q7: "What are the possible results of an angiogram?"

Q8: "Does age have an effect on the criticality of colon cancer?"

MU answers these questions by traversing the relations between actions and nodes "higher" in the inference network. For example, Q6 is answered by finding all the nodes for which EKG provides evidence. The planner may ask either for the *immediate* consequence of knowing EKG, or for the consequences to any desired depth of inference.

Focus questions help a planner establish focus of attention. For example:

Q9: "Give me all diseases that are triggered and dangerous."

Q10: "What are all of the critical diseases for which I have no information?"

Q11: "Are any hypotheses confirmed?"

Questions about **multiple effects** allow the planner to combine the previous question types into more complex queries such as "What tests can discriminate between angina and esophageal spasm?" In this case, the term *discriminate* is defined to mean "simultaneously increase the level of belief in one disease and lower it in an other."

Hypothetical questions allow the planner to identify dependencies among actions. For example, one can ask, "Suppose the response to trial therapy is positive. Now, could a stress test still have any bearing on my belief in angina?"

With the ability to define control features and answer such questions, we quickly reimplemented MUM's strategic phase planner. Most of the effort went into adding declarative definitions of control features and their combining functions to MUM's medical inference network.

9 Conclusion

MU supports the construction of systems that have the characteristics of prospective reasoning identified in Section 2: Prospective reasoning involves answering the question, "What shall I do next," given uncertainty about the state of the world, the effects of actions, tradeoffs between the costs and benefits of actions, and precondition relations between actions. The six classes of questions, discussed above, help planners to decide on courses of action despite uncertainty. Questions about **state** make uncertainty about hypotheses explicit. **Hypothetical** questions and questions about **effects** make uncertainty about the outcomes of actions explicit. Questions about **goals** and **multiple effects** help a planner identify the tradeoffs between actions. And **hypothetical** questions make dependencies between actions explicit.

We are currently extending MU's abilities in several ways. One project seeks to automate the process of acquiring strategies. It attempts to infer strategies from cases, asking the expert to supply new control features when the current set is insufficient to represent the conditions under which strategies are appropriate. We are also building an interface to help acquire combining functions. This task becomes confusing for the expert and knowledge engineer alike when levels of belief must be specified for combinations of many data. We discuss related work on the design of functions to extrapolate from user-specified combining functions in (Cohen, Shafer and Shenoy, 1987). A third project is to implement sensitivity analysis in MU. The goal is to add a seventh class of queries, of the form, "To which data and/or intermediate conclusions is my current level of belief in a hypothesis most sensitive." This will facilitate prospective reasoning by giving the planner a dynamic picture not only of its belief in hypotheses, but also in its *confidence* in these beliefs. With sensitivity analysis the prospective reasoner will be able to find weak spots in its edifice of inferences and shore them up (or let them collapse) before they become the basis of unwarranted conclusions.

Chapter 4

Prospective Decision Making

1 Introduction

Making a decision is the process of accumulating and integrating evidence, selecting an alternative, and perhaps acting on that selection. The first step is typically performed by a decision analyst and a client in a relationship similar to that of a knowledge engineer and an expert. It involves building a complete *model* of the decision. Simple algorithms can then select an alternative that is optimal with respect to the model. The model must provide a way to evaluate the worth of alternatives, and this typically involves constructing a search space of outcomes of the alternatives. If the model is to faithfully represent the real world, then this space can be combinatorially large. By AI standards, searching it is relatively inexpensive, but the cost of *constructing* the space can be enormous. This is due to two related reasons. First, it is done by people, not computers, and involves the same kind of ponderous, error-ridden interviewing that characterizes the knowledge acquisition bottleneck in knowledge engineering. Second, because constructing the decision is accomplished before selecting an alternative, the selection process provides no guidance to the construction process and, worse, the constructed model includes (at great expense) information that is not relevant to the selection process.

This report presents an alternative approach called *constructive decision making* that merges the construction and selection processes. It iteratively asks whether more information could increase confidence in a decision and, if so, decides what information is needed. It views decision-making as a transition through *decision states*, each of which represents a decision supported by successively more evidence. Thus, the algorithm can offer a decision at any time, with the proviso that more evidence might result in a better decision.

Constructive decision making underlies a decision support system that incrementally identifies the factors that influence a decision, and moves from states where alternatives have weak support to states in which choices are more clear. This system,

called CDM, emphasizes the process of acquiring and structuring just the information required for a decision. It characterizes the current state of the decision with respect to strength of support for each alternative. If no alternative is clearly superior, it seeks information about factors of the decision that can discriminate the alternatives. It never requests information that it does not need to develop the decision; that is, it never asks for irrelevant information. The process of acquiring information about attributes continues until a decision is forced or a clear choice emerges.

Since constructive decision making merges the processes of constructing a decision and selecting an alternative, it is ideally suited to AI programs that must construct decisions for themselves. This situation arises in domains where the relevance of factors cannot be determined until run-time; that is, domains where decision analysts cannot construct a combinatorial model of a decision ahead of time. For example, programs with dynamic control structures must construct dozens or hundreds of control decisions based on factors whose relevance changes in the course of problem-solving (HEARSAY, MUM). Moreover, in real-time control problems, the constructive decision making approach can offer satisficing decisions that are the best possible given the time available to collect and process evidence.

1.1 Why Not Just Rely on the Decision Sciences?

The development of decision theory was motivated by a desire to produce optimal decisions in difficult or complex problems. Decision theory has become the theoretical basis for most other work in decision making because it provides a consistent, rigorous mathematical approach to representing decision problems under uncertainty so that they can be examined (North 1968), and offers a definition of optimality that relies on the state of knowledge (Luce and Raiffa, 1957).

Decision analysis provides a methodology of decision making based on decision theory (von Holstein 1971). Decision analysts use formal procedures to solve decision problems by balancing different factors of a decision (Howard 1966). It is a comprehensive model of decisions that provides the ability to place a dollar value on uncertainty, justify major decisions, and encode subjective information about risk aversion and uncertainty (Howard 1966, Raiffa 1970).

However, decision analysis is an expensive, time consuming and painstaking process. The analyst's task is far from easy. Utility assessments force people to place a monetary value on distinctly non-monetary outcomes such as environmental damage or an education (Howard 1968). Probability assessments are never clear-cut and are often difficult to obtain (Raiffa 1970). Moreover, before a decision can be analyzed, the alternatives, and their outcomes, probabilities, and utilities must already be specified. Decision analysis does not easily accommodate the notion of constructing decisions by acquiring information only as needed.

The primary decision representation, a decision tree, becomes a "bushy mess" for

problems involving more than a few alternatives (Raiffa 1970). Consequently, the representation may be difficult to construct, manage and analyze for complicated problems.

Skilled decision analysts are able to reduce this complexity by narrowing the number of distinct alternatives considered for each action (e.g. Henrion and Cooley 1987), but this simply illustrates the need for further assistance in managing the complexity of the task.

Decision support systems assist humans in solving unstructured problems by providing models and data (Ford 1985). Most standard decision support is model-based, designed for specific areas, such as corporate planning, portfolio management, and marketing, using mathematical models of decision making expertise (Wang and Courtney 1984). These systems assist decision making by providing previously described simulations of possible repercussions of a decision selection in tightly constrained domains. Consequently, model based decision support reduces the informational demands on the user, but doesn't describe how to build the representation or make the selection.

A few systems have been developed to assist the general task of making decision selections. ARIADNE, Alternative Ranking Interactive Aid based on DomiNance structural information Elicitation, addresses the problem of eliciting from a user a dominance structure for selecting from multiple criteria alternatives (Sage and White 1984). Leal and Pearl describe a program that interacts with the user to construct decision trees (Leal and Pearl 1977). GODDESS, a GOal-Directed DEcision Structuring System, produces a hierarchical goal representation of decision alternatives by selectively focusing the user's attention on the most crucial issues (Pearl et al. 1982).

Both these general-purpose systems focus on constructing a representation of the decision that will facilitate selection. Once constructed, the evaluation should be straightforward. The traditional split between building and resolving decisions exists to ensure optimal decisions. First, the decision is built, defining a search space. Then, it is exhaustively searched, ensuring the best decision.

Many decisions must be made in dynamic situations by intelligent programs without human intervention. In these cases, the program must define the decision by itself. Because we wish to avoid combinatorial search spaces and promote real-time problem solving, the current two-step build-and-resolve view of decision making must be replaced by a dynamic, constructive algorithm. This algorithm must:

1. not hold up processing while waiting for evidence
2. not consume resources building a combinatorial model that must be searched exhaustively
3. provide the 'best' decision at any point in processing, but in supporting this capability will sacrifice the goal of optimality
4. work even when the alternatives and attributes are not known or specified a priori, but emerge in the course of decision making.

These capabilities are beyond those supported by traditional decision science. The decision process must respond to a dynamic environment in which precise information is not always available and conclusions may be drawn with varying confidence in less time. Consequently, dynamic planning is predicated on qualitative assessments with graceful degradation of the decision process under conditions of less available data and time.

1.2 How Do People Decide?

Humans have the decision making capabilities that we would like to include in our intelligent programs. Kahneman and Tversky and many of their collegues (Kahneman, Slovic and Tversky 1982, Payne 1982) performed experiments to investigate these capabilities. They observed that because people have limited processing capabilities they tend to rely on heuristics to assess probabilities and predict outcomes and that these heuristics sometimes lead to difficulties.

These heuristic strategies often result in *satisficing* in which global rationality (optimizing) is replaced by a definition that is more compatible with human capabilities (Simon 1981). Satisficing bypasses some of the difficulties of evaluating utility and comparing incommensurable attributes and limits the amount of search which needs to be done (Coombs, Dawes and Tversky 1970).

A technique for further investigating human decision making heuristics is protocol studies of people making complex decisions.

- Analysis of four senior auditors analyzing an audit case showed that in the course of the decision process, the decision making operations of structuring, search and analysis was intermixed, and that evaluative criteria emerged as the decision progressed. (Biggs and Mock 1983)

- Payne, Braunstein and Carroll (Payne 1978) found that subjects eliminate alternatives by deleting any whose attributes don't equal a criterion value and compare pairwise the remaining alternatives.

- In protocol studies of people selecting a house, Ola Svenson (1979) found that attributes varied across subjects and became more specific and detailed as the decision process progressed. Additionally, new attributes tend to emerge in the course of making the decision.

- Bettman and Jacoby found that subjects used a strategy called "choice by feedback processing" in which they alternated between considering alternatives with respect to attributes and attributes with respect to alternatives (Svenson 1979).

The most compelling facets of these studies are that people successfully use heuristics, often tailored to pairwise comparisons, to reduce the computational requirements

of complex decision tasks and that decision making is a constructive process that selectively includes, analyzes and discards information as it becomes more or less relevant to the task. Consequently, if we wish to reduce the combinatorial search space of decisions and produce real-time decision making behavior, these heuristic methods provide some suggestions of how to do so.

2 Constructive Decision Making

The development of constructive decision making (CDM) was motivated by the need for intelligent programs to define and evaluate decisions autonomously. By relying on a model that contains aspects of human decision making, it addresses some of the problems in decision analysis.

The research presented here focuses on these problems: comparing incomparable attributes and constructing the solution to a decision problem. Comparing incomparable attributes refers to the problem of collapsing all relevant factors of a decision into a single measure, as in utility theory and more so in probability. Mapping all factors onto a single scale is difficult to do when no scale is obvious or the factors do not obviously fit the scale, and is difficult to interpret because possible uncertainty has been factored out. Constructive decision making is qualitative; it emphasizes differences over absolute judgments, and does not collapse all factors to a single scale. The approach is constructive because all necessary information may not be available initially, and in fact, may not all be needed. The remainder of this report will describe this methodology for solving two alternative, multiple attribute decision problems.

2.1 Decision Typology

The core of constructive decision making is the decision typology. The typology characterizes decision situations using domain independent dimensions, guides the collection of support and provides the best possible decision given available evidence when there is some distinction between the alternatives. The goals of decision making as performed by the typology are:

- to select an alternative that seems reasonable given the available information
- to reduce uncertainty in the selection of the alternative by collecting evidence to support it or refute it.

We developed the typology to model the process of making decisions based on incomparable attributes. We call problems of this type *apples and oranges problems.* When you compare apples and oranges in a grocery store you may find one fruit preferred on the basis of flavor and the other on the basis of quality. If you could combine the attributes to compare the alternatives on a single, composite attribute, then the

choice is often clear. But if, as in this case, flavor and quality are not easily combined, then the choice between apples and oranges is problematic. The description of the typology will refer back to this example.

Decision States

We begin the discussion with simple 2-alternative, 2-attribute problems typified by the *apples and oranges problem.* Decision alternatives are compared on their salient attributes.[1] A *decision state* in our model is a concise statement of the current combination of salient attributes, including how well the alternatives are distinguished on the available attributes and how important those attributes are. We identified five *dimensions* that describe decision states.

The dimensions in the decision typology are abstracted from the decision situation and are used to reason about the state of the decision. Decision problems between two alternatives can be characterized along four dimensions:

- significant difference
- conflict
- order
- greater-than.

Significant difference with respect to an attribute indicates whether the values of the attribute for the two alternatives are distinct. Assuming that the values of all the other attributes for the two alternatives are equal, can a decision be made based only on *this* attribute? This dimension determines whether the difference between two alternatives is significant enough to support a decision, which effectively avoids the issue of exactly what value each alternative has or what distribution of values can be expected and how much difference is required to be significant. The formal definition[2] used in the typology is:

$$Sd[A_i] = \begin{cases} 1 & \text{if } A_i[p] \text{ and } A_i[q] \text{ are distinct} \\ 0 & \text{otherwise} \end{cases}$$

Otherwise indicates no significant difference or that we lack evidence to tell whether there is a significant difference.

[1] Throughout this report "attribute" is used loosely to refer to features of alternatives that are salient to the task of selecting the best alternative. This definition is vague enough to accommodate *outcomes*, *goals* or *characteristics*. The distinction will be refined in Section 2.1.

[2] In the descriptions that follow, alternatives are referred to as p and q, attributes as A_i and A_j, and values of attributes for specific alternatives as $A_i[p]$. The symbols $\succsim$ and $\precsim$ indicates preference between two values, and $>$ and $<$ have their normal meanings.

A conflict exists when the two attributes support different alternatives, that is, we have conflicting evidence. Formally, this is described as:

$$C[A_i, A_j] = \begin{cases} 1 & \text{if } A_i[p] \tilde{>} A_i[q] \text{ and } A_j[p] \tilde{<} A_j[q] \text{ or} \\ & \text{if } A_i[p] \tilde{<} A_i[q] \text{ and } A_j[p] \tilde{>} A_j[q] \\ 0 & \text{otherwise} \end{cases}$$

Importance indicates whether one attribute is considered more influential than the other. Once again we have avoided the issue of *why* we believe this. It may be because the attribute itself is more important, independent of the values of the alternatives on the attribute, or that the values of the alternatives are so radically different on a particular attribute that it is more discriminating that the others.

$$I[A_i, A_j] = \begin{cases} 0 & \text{if importance}(A_i) = \text{importance}(A_j) \\ ? & \text{if relative importance is unknown} \\ 1 & \text{if importance}(A_i) > \text{importance}(A_j) \\ & \text{or importance}(A_i) < \text{importance}(A_j) \end{cases}$$

Greater_than is simply an extension of Importance to indicate *which* attribute is more important if, in fact, one is.

$$\tilde{>}[A_i, A_j] = \begin{cases} 0 & \text{or importance}(A_i) < \text{importance}(A_j) \\ 1 & \text{if importance}(A_i) > \text{importance}(A_j) \end{cases}$$

These four dimensions are the basis of the five that describe a decision state. Since a decision state describes a comparison between alternatives on two attributes, the state description includes the significant difference evaluation on two attributes, A_i and A_j, not just one as shown in the definitions.

These dimensions can be illustrated in the context of the problem of selecting fruit: p is apples, q is oranges, A_i is *quality* and A_j is *flavor*. If the quality of apples is "good" and the quality of oranges is "poor," then $Sd[quality] = 1$ because good and poor are distinct values. Similarly, if one prefers the flavor of oranges to that of apples then $Sd[flavor] = 1$. Since apples have better quality but oranges taste better, $C[quality, flavor] = 1$. Finally, if quality is preferred to taste $I[quality, flavor] = 1$ and $\tilde{>}[quality, flavor] = 1$. These dimension values can be put together to form a vector descriptor of the state of the decision represented as [1, 1, 1, 1, 1].

Actions

The purpose of the characterization of the decision on these dimensions is to reason about how to develop the decision dynamically. Decision making is viewed as an constructive process. Five general actions apply in the multi-attribute, two-alternative model to help construct a decision:

- decision
- transformation by attribute
- transformation by importance
- substitution
- combination

Decision means selecting an alternative based on available evidence. For example, if importance distinguishes two attributes ($I[A_i, A_j] = 1$), then the alternative favored by the more important attribute is the decision. A decision can always be made, but with varying confidence. If you wish to increase confidence, another action should be performed instead.

The action **transformation by attribute** (abbreviated *Ta*) seeks to transform the current decision state by gathering information about one of the attributes. If the current information about an attribute is uncertain or unknown, this action attempts to resolve that uncertainty. The intent of a transformation is to change one decision state into another, hopefully more facilitative state. However, the desired transformation to a better state may not be possible; the actual transformation depends on the evidence obtained. For example, we may gather evidence about A_j with the hope of getting additional support for the currently favored alternative (supported by information we already have about A_i), but if the evidence, when obtained, indicates that A_i and A_j actually support different alternatives, then we end up in a state with a conflict.

The action **transformation by importance** (*Ti*) is like transformation by attribute but involves obtaining importance information.

We may wish to include other attributes. Two actions, substitution and combination, add new attributes. When one of the two existing attributes doesn't provide a significant difference, a new attribute may be **substituted** (*Su*) for it by discarding the existing insignificant attribute and replacing it with a new one.

Alternatively, we could **combine** (*Co*) the new attribute with the existing ones by taking advantage of the fact that there are only two alternatives. Since an attribute may only support one or the other of the alternatives, the attributes may be *clustered* together according to which alternative they support. Clustering is the key to extending a basic two-alternative, two-attribute representation to two-alternative, N-attribute representation, and finally to an M-alternative, N-attribute representation, because it permits complex decision situations to be constructed iteratively within the framework of the decision typology.

These actions describe the state transitions that gather information and judgments about a decision and structure them such that the selection of an alternative becomes relatively obvious.

Typology

Considering all the possible combinations of the values of the five dimensions and pruning out isomorphic states (isomorphic with respect to the actions that may be taken) produces 23 basic states. Basic states have no clustered attributes.

The 23 states can be arranged in a table (Figure 4.1). The illustrative apples and oranges problem discussed earlier is state 23 in this table. In English, state 23 says:

> For p=apples and q=oranges, the strength of evidence for *quality*, A_i[p] and A_i[q], is sufficient to claim that the difference supports a choice between p and q. The strength of evidence for *flavor*, A_j[p] and A_j[q], is sufficient to claim that the difference supports a choice between p and q. There is a conflict between p and q on A_i and A_j, and the attribute A_i is more important than A_j.

Isomorphic states have been pruned out of the table. A full table would include 40 states. From the perspective of how a decision-maker acts, the 40 decision states contain some redundancies. Consider these states:

State 18a: $S[A_i] = 1, S[A_j] = 0, C[A_i, A_j] = 1, A_i \tilde{>} A_j$
State 18: $S[A_i] = 0, S[A_j] = 1, C[A_i, A_j] = 1, A_j \tilde{>} A_i$

In English, this state means

> "The dimension for which your evidence supports a decision is the most important dimension."

The states are identical in the sense that a decision-maker would not act differently in response to them. Consequently, the two states are represented only by state 18 in the table.

Once the state of a decision has been identified, the table can be used to look up the set of possible actions. Figure 4.1 shows the appropriate actions for each state. The actions are divided into two rows. The first shows the actions for states with complete evidence. The second describes actions to be performed when some of the state information is missing.

Each of the actions has well-defined transitions determined by the new information that they gather. Transformation is an appropriate action for any decision state with 0 in either of its first two rows or ? in its fourth. Substitution is appropriate when two alternatives are not differentiated on an attribute ($Sd[A_i] = 0$); since the attribute does not distinguish the alternatives, it should be replaced with one that does. Combination is appropriate anytime the decision is uncertain and more evidence should be gathered. This is typified most by states in which attributes support different alternatives (there is conflict) and attributes have equal importance. The most appropriate actions (not all the possible actions) for a given state are listed in the table with numbers that

State #		0	1	2	3	4	5	6	7
Sd[A_i]		0	1	1	0	1	1	0	0
Sd[A_j]		0	0	1	0	0	1	0	1
C[A_i, A_j]		0	0	0	1	1	1	0	0
I[A_i, A_j]		?	?	?	?	?	?	0	0
$\tilde{>}$[A_i, A_j]		*	*	*	*	*	*	*	*
Actions	All Info	Co D	Su D	D		Su,Co D	Co	Co D	Su D
	Part Info	Ta 0,1,4 Ti 6, 12,20	Ta 1,5,8 Ti 7, 13,14	Ti 5,8 21	Ta 3,4,5 Ti 9, 16,22	Ta 2,4 Ti 10, 17,18	Ti 11, 19,23	Ta 6, 7,10	Ta 7, 8,11

State #		8	9	10	11	12	13	14	15
Sd[A_i]		1	0	1	1	0	1	0	1
Sd[A_j]		1	0	0	1	0	0	1	1
C[A_i, A_j]		0	1	1	1	0	0	0	0
I[A_i, A_j]		0	0	0	0	1	1	1	1
$\tilde{>}$[A_i, A_j]		*	*	*	*	0	0	0	0
Actions	All Info	Co D	Su Co	Su Co	Co	Co Su	Su D	Su Co	Co D
	Part Info		Ta 9,10,7	Ta 10,11,8		Ta 12,13, 14,17,18	Ta 13,15, 17,19	Ta 14,15, 18,19	

State #		16	17	18	19	20	21	22	23
Sd[A_i]		0	1	0	1	0	1	0	1
Sd[A_j]		0	0	1	1	0	1	0	1
C[A_i, A_j]		1	1	1	1	0	0	1	1
I[A_i, A_j]		1	1	1	1	1	1	1	1
$\tilde{>}$[A_i, A_j]		0	0	0	0	1	1	1	1
Actions	All Info	Co Su	Su Co	Su Co	Co	Co Su	Co D	Co Su	Co
	Part Info	Ta 16,17, 13,14,18	Ta 17,19, 15	Ta 18,19, 15		Ta 20,13, 14,17,18		Ta 22,13, 14,17,18	

Figure 4.1: Basic Decision Typology with states and actions
State numbers appear at the top of each column. For each state, the columns contain the values for the five dimensions followed by the actions that seemed the most reasonable. The actions are partitioned according to whether all the information has been acquired for the dimensions. For example, if only partial information has been acquired, then transformations are better. The numbers following some of the actions refer to the state transitions that may occur if that action is taken.

indicate the set of possible destinations if the actions is performed. Note, as mentioned earlier, it is not possible to say exactly which of these states will arise until after the action is performed.

Effect of Actions on Decision State

Adding a new attribute via substitution and combination potentially affects every cell in a decision state, that is, each value $\mathrm{Sd}[A_i]$, $\mathrm{Sd}[A_j]$, $\mathrm{C}[A_i, A_j]$, $\mathrm{I}[A_i, A_j]$, and $\tilde{>}[A_i, A_j]$. In combination with a new attribute, a previously insignificant single attribute may form part of a significant cluster (e.g., $Sd[A_i] = 0$ but $Sd[A_i \& A_{new}] = 1$). $\mathrm{C}[A_i, A_j]$ may change if the new attribute produces a conflict, and $\mathrm{I}[A_i, A_j]$ and $\tilde{>}[A_i, A_j]$ change simply by clustering attributes. Within the framework of our typology, the effects of adding a new attribute are:

1. to introduce a conflict where there was none
2. to take a side in a conflict
3. to join the consensus ($\mathrm{C}[A_i \& A_j, A_{new}] = 0$) but lend it legitimacy since $Sd[A_{new}] = 1$
4. to introduce an ordering where there was none
   (e.g. $\mathrm{I}[A_i, A_j] = 0$ but $\mathrm{I}[A_i, (A_j \& A_{new})] = 1$)
5. to change an ordering (e.g., $\tilde{>}[A_i, A_j] = 1$ but
   $\tilde{>}[A_i, (A_j \& A_{new})] = 0$

Figure 4.2 shows all the possible actions and their effects for a single state in the typology, state 4. In this example, there is enough of a difference to support a decision on A_i, but not A_j, and the evidence of the two attributes is contradictory. Four actions are appropriate: transformation by attribute (the 0 value for $\mathrm{Sd}[A_j]$ may indicate insufficient evidence), transformation by importance, substitution (for A_j), and combination. Note that it is possible to return to the same state, state 4, but by different paths. Substituting A_j or combining attributes transforms state 4 to state 5. But note that when state 5 was reached by combining attributes, one of them, A_i or A_j, actually represents the evidence of two attributes and so supports a decision more strongly.

Expanding the Definition of Attribute As presented in the typology, attributes refer to features of alternatives that are salient to the task of selecting the best alternative. This definition accommodates *outcomes*, *goals* or *characteristics*. Because each affects the decision differently and because we would like to be able to reason about the interaction of goals, the model has been extended to account for these separate types of attributes and how they interact.

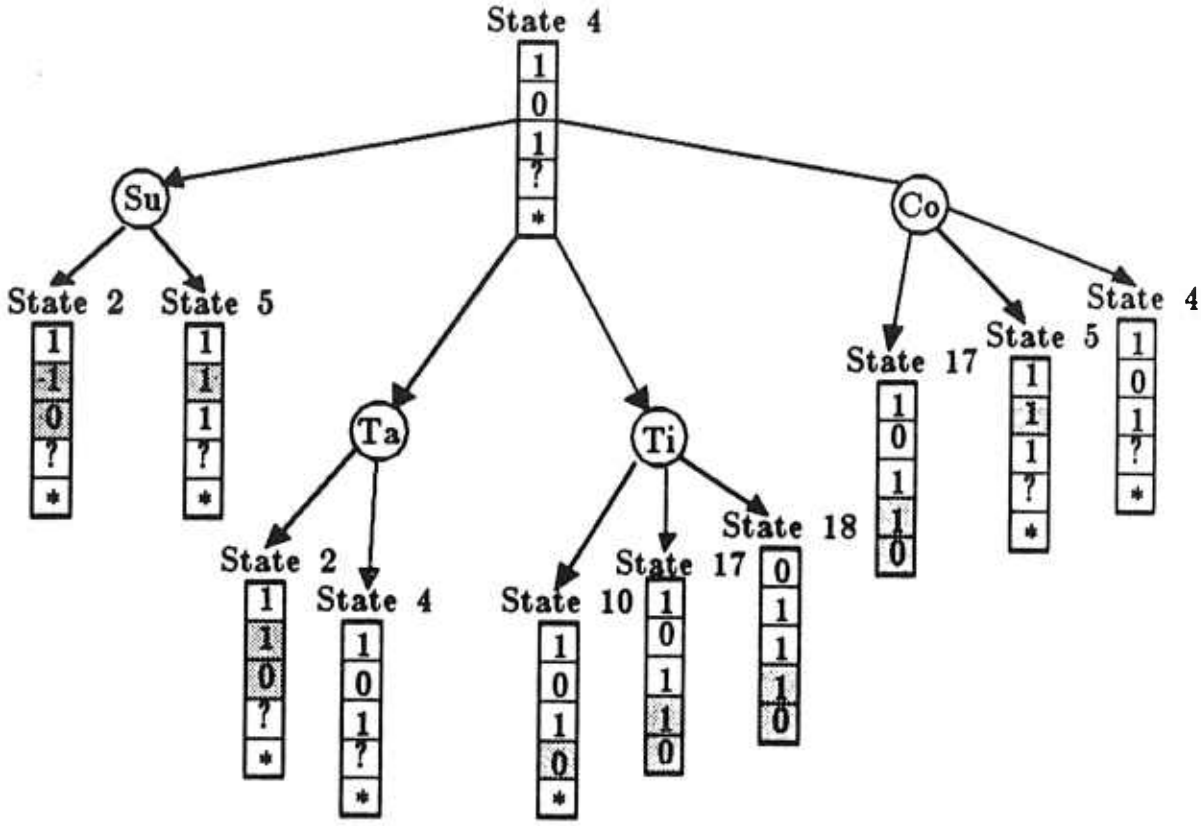


Figure 4.2: Single Transition with Multiple Attributes

Goal is the desired result (outcome) of a decision along a particular attribute.

Attribute is the actual result or repercussion of a decision, e.g., when you purchase a car, one result may be gas mileage, another may be reliability.

Expected outcomes are the expected results, with respect to a goal, of making a particular decision.

Characteristics are the actual attributes of the decision alternatives that contribute to the performance of those alternatives on each of the goals.

As an example, imagine the decision being between two cars, a Porsche 928 and a Nissan Maxima. Two of my goals in selecting a car may be fast acceleration and great gas mileage. Given the two choices, the actual attributes of the decision might be fast acceleration for the Porsche and reasonable gas mileage for the Nissan. The expected outcomes are that the Porsche will be better on acceleration and the Nissan will be better on gas mileage. A characteristic of the car that is related to both attributes is engine size. So, the smaller engine size of the Nissan gives it better gas mileage, but slower acceleration. Knowing that a characteristic supports one goal and detracts from another allows us to recognize a trade-off and attempt to seek other evidence that will distill its effect.

To summarize, these four are related in the following way:

- A *decision* has many *attributes.*
- These *attributes* provide evidential support (by clustering as described earlier) to one of the two alternatives.
- A *goal* is a desired outcome along a particular *attribute.*
- Desired outcomes may be compared to *expected outcomes* on each attribute.
- An *expected outcome* for each alternative is a function of some subset of the set of *characteristics.* So expected outcomes form clusters of characteristics that lend support to the claim that a particular alternative performs better on a particular goal.

As can be inferred from their relationships described above, the four terms form a hierarchy (see Figure 4.3).

3 How Does it Work? Program Implementation

A decision support system has been built based on the typology (Howe 1986b). It asks the user general questions about his/her decision, constructs a representation (as

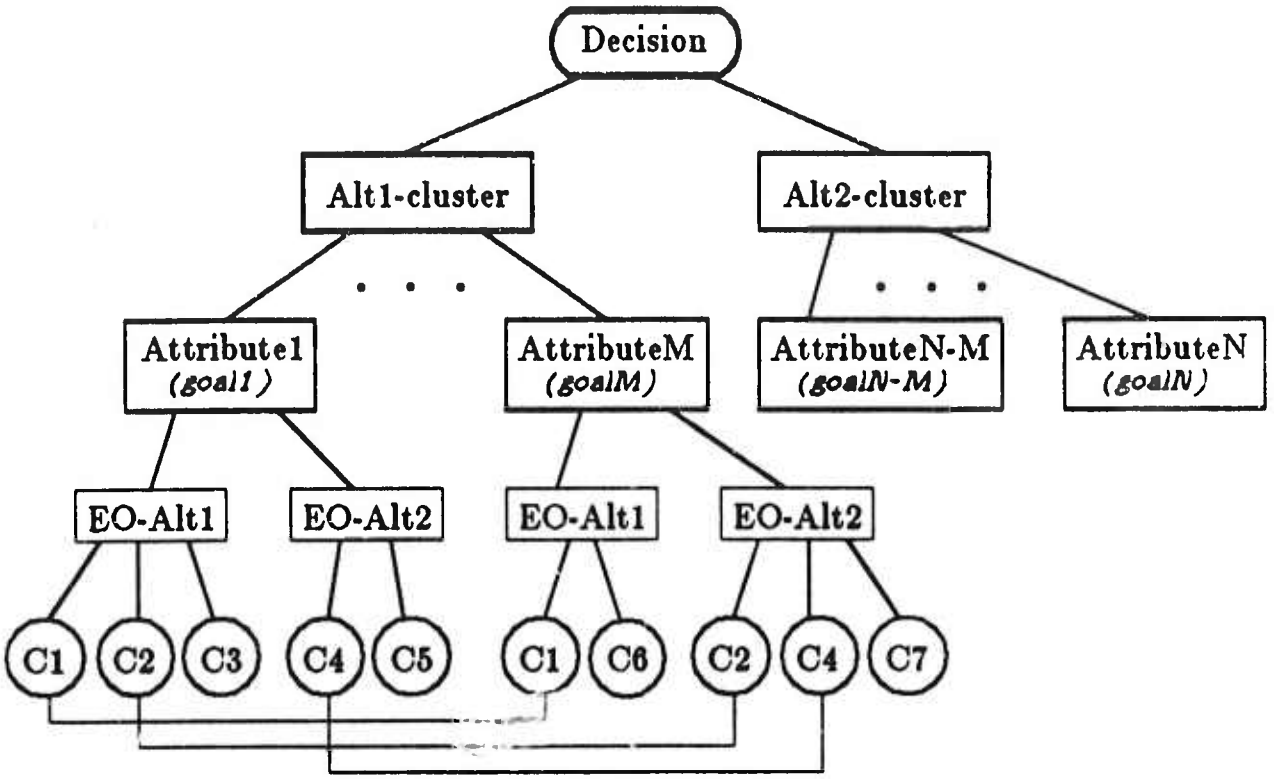

Decision
Alt1-cluster
Alt2-cluster
Attribute1
(goal1)
AttributeM
(goalM)
AttributeN-M
(goalN-M)
AttributeN
(goalN)
EO-Alt1
EO-Alt2
EO-Alt1
EO-Alt2
C1
C2
C3
C4
C5
C1
C6
C2
C4
C7

FIGURE 4.3

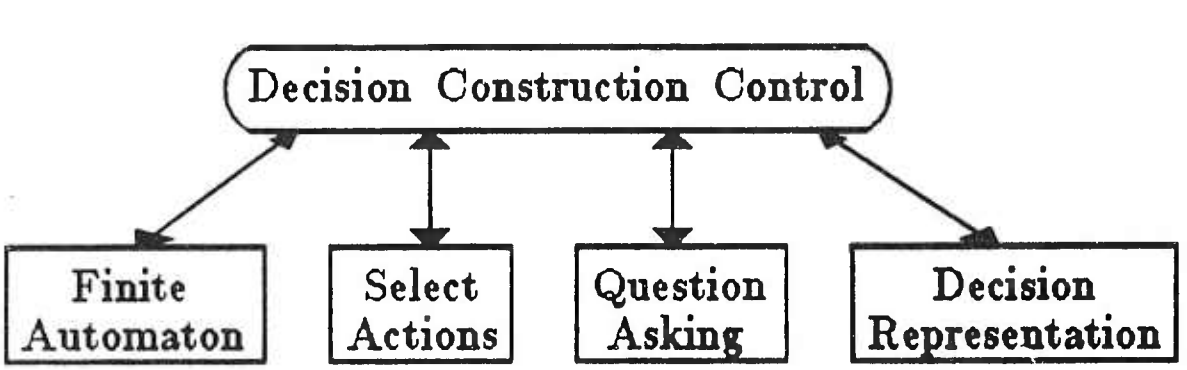

Decision Construction Control
Finite Automaton
Select Actions
Question Asking
Decision Representation

Figure 4.4:

described in the last section), and ultimately suggests an alternative as a decision. The possible actions and possible decisions, along with reasons, for each state are suggested by the typology.

A CDM based decision support system requires four parts: a finite automaton to control state transitions, rules for selecting actions at each decision state, a question-asking interface, and an internal decision representation. These parts interact via a controlling program as shown in Figure 4.4. With respect to implementing the typology, the last two modules, question-asking and internal representation, are fairly simple. The primary role of the question-asking module is to translate the actions to a form that the user can understand – direct questions that request the necessary information. This is implemented simply by mapping the actions to pre-determined questions. The internal decision representation forms the database for the decision information as it is gathered and provides routines to print and explain the decision being constructed. The first two modules implement the decision typology, and are described below.

3.1 Finite Automaton

The decision typology is represented as a finite automaton in which the states are the decision states described earlier and the arcs are labeled with the decision actions. The finite automaton module translates the dimensions of the typology into a state representation, determines the applicable actions and transitions for the state, and creates new states, as necessary.

Performing an action causes a state transition. The program determines what actions are applicable and predicts their possible results. For example, if we start from state 4 as in Figure 4.2, the applicable actions are Su, Co, Ta, and To. These actions may result in states 2, 4, 5, 10, 17, and 18. The set of applicable actions for a given state are determined by following all applicable rules presented in Figure 4.5.

Each action has a set of rules for determining the possible changes to the dimensions and so the possible destination states. For transformation by attribute (Ta), getting attribute information causes the dimension Sd and possibly the dimension C to change. In transformation by importance (Ti), new states include changes to Importance and $\tilde{>}$. Because most of the dimensions are determined relative to the attributes, performing a substitution changes Sd of the attribute being substituted, C, I and $\tilde{>}$. Combination actions conceivably change every dimension (in particular ways) because the new attribute gets combined with the existing ones.

Decisions can be made, if necessary, by accumulating supportive evidence. This evidence is gathered from the following rules which propose a selection and explain why.

1. If A_i has Sd=1 and A_j has Sd=0, then one can decide based on A_i.
2. If A_j has Sd=1 and A_i has Sd=0, then one can decide based on A_j.

1. If the information about an attribute is unknown or uncertain (e.g., Sd=0), then suggest Transformation by Attribute.

2. If the relative importance of the attributes is unknown (e.g., I=?), then suggest Transformation by Importance.

3. If an attribute doesn't provide adequate support (e.g., Sd=0), then suggest Substitution.

4. If both attributes provide significant, corroborating evidence (e.g., $Sd(A_i)=Sd(A_j)=1$ and $C(A_i, A_j)=0$), then suggest Decision.

5. If at least one attribute provides significant evidence which doesn't conflict with the other attribute and it is considered to be more important (e.g., $Sd(A_i)=1$, $C(A_i, A_j)=0$, and $\tilde{>}(A_i, A_j)=1$), then suggest Decision.

6. Suggest Combination anytime.

Figure 4.5: Rules for determining possible actions

3. If there is no conflict, then one can decide based on A_i.

4. If A_i is more important than A_j, then one can decide based on A_i.

5. If A_j is more important than A_i, then one can decide based on A_j.

3.2 Action Selection

Once we know what the possible actions are, we must select one. The Action Selection module's function is to select exactly which action should be implemented in a given situation.

Ideally, selecting the action should rely on domain dependent information or strategies promoted by the user. For example, a possible action may be to transform by attribute. In most cases, this would be the preferred action because it provides the most evidence; however, if it is expensive to get that information, then it may be preferable to do something else. The current implementation does not include domain dependent information but relies instead on a simple, heuristic, though possibly ad hoc ordering. Sometime later, this ordering could be replaced or superseded by a rule base and interpreter or some other general representation of domain information.

To augment the simple ordering scheme, the program does some rudimentary reasoning about the possible destinations of an action. At least one of the possible destinations must be a state in which a decision can be made, hence a change for the better. Otherwise some other action will be performed.

3.3 Construction Control

The construction control module controls the other modules and governs their interaction. It determines the possible actions given the current state, selects the most appropriate, takes the action by asking the user questions, determines when its appropriate to finish, updates the representation and allows the user to examine the evolving decision. Most of this simply involves calls to modules already described. However, because it forms the interface between the representation and the user, interpretation of the users responses and requests is crucial to determining when all the applicable information has been accumulated.

3.4 Results

The first version of the CDM program has been tested by several people. The user is presented with a decision and some facts about it. They were to use the system to make a decision between two alternatives. The following transcript is an annotated version of one of those sessions. The user was asked to select between two houses described in a real estate advertisement.

The following is a portion of a transcript of a decision making session using the decision typology. The system's responses are in `typewriter font`, the user's responses are in roman font, and annotations are in *italics font* for ease of identification. Portions that have been omitted without an explanation are indicated by vertical ellipsis. After every system action, the user may view parts of the growing representation. This option is presented after every action in a menu. Often that section of the transcript has been omitted to save space.

```
Lisp> (construct-decision)

Welcome to Version 1.0 of the Constructive Decision Support System.

The purpose of this program is to help you make a decision by asking you
questions about the alternatives, keeping track of your answers, and
trying to focus on the salient aspects of it. The system builds a sort
of pro/con representation to help explain why one alternative seems best.

The current version has a few restrictions:
    - the decision involves exactly two alternatives
    - all goals require at least two characteristics
    - once the program has selected an alternative, you can't override it

The program will ask you to evaluate characteristics and goals of your
```

decision. In answering these questions, you should consider ONLY the feature that you're being asked about. Don't evaluate the features with the effects of others in mind. Try to select the features and think of them independently of one another.

Finally, the program will ask you questions that refer to alternatives, goals, characteristics, values and importance.

Would you like an explanation of these terms? y

At this point, the program gives definitions of the basic terms: alternatives, goals, characteristics, values, and importance as they are described in the paper.

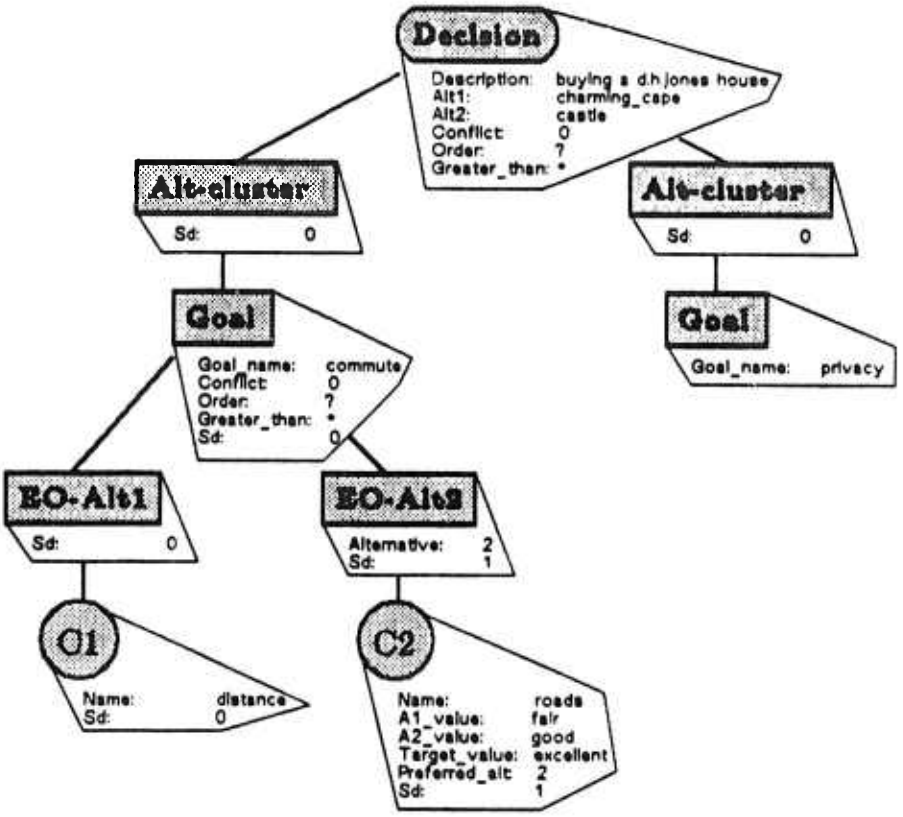
Please give a short text description of the decision problem:
[empty line to end] buying a d.h.jones house

What is the name of the first alternative? [one word] charming_cape
What is the name of the second alternative? [one word] castle
What is a goal of this decision? [one word] commute
What is another goal of this decision? [one word] privacy
What is a characteristic that affects the performance of the goal Commute? [one word] distance
What is another characteristic that affects the performance of the goal Commute? [one word] roads

Do you have information on the values of the alternatives on the feature, Roads? [y or n] y
What is the value of Charming_Cape for characteristic Roads? fair
What is the value of Castle for characteristic Roads? good
What is the best value that they can have? excellent
Is the difference between Charming_Cape and Castle significant on characteristic Roads? yes
Which alternative performs better on roads?
[CHARMING_CAPE = 1 & CASTLE = 2, 0 = neither] 2

To examine all or part of the decision being constructed,
select one of the following options:
1 Print the decision tree
2 Print a goal tree
3 Explain the current state
4 Break to lisp temporarily [type (continue) to return when finished]
5 continue with the program

which one? 1


Decision
Description: buying a d.h.jones house
Alt1: charming_cape
Alt2: castle
Conflict: 0
Order: ?
Greater_than: *
Alt-cluster
Sd: 0
Alt-cluster
Sd: 0
Goal
Goal_name: commute
Conflict: 0
Order: ?
Greater_than: *
Sd: 0
Goal
Goal_name: privacy
EO-Alt1
Sd: 0
EO-Alt2
Alternative: 2
Sd: 1
C1
Name: distance
Sd: 0
C2
Name: roads
A1_value: fair
A2_value: good
Target_value: excellent
Preferred_alt: 2
Sd: 1

The user has entered the basic information needed to start the system. The program used this information to build the tree displayed above by the user.

```
Do you have information on the values of the alternatives on
   the feature, Distance?  [y or n] y
What is the value of Charming_Cape for characteristic Distance? half_hour
What is the value of Castle for characteristic Distance? half_hour
What is the best value that they can have? 10_mins
```

The system just performed a transformation by feature. It had no information about the feature, Distance, and so asked the user. In the section of the transcript that has been omitted here, the system performed a tranformation by order to determine which feature, Distance or Roads, is more important. Action selection is performed conservatively. Importance information is requested because it provides evidence used to distinguish the currently available characteristics should it happen that they are the only ones available. If other characteristics get included, the importance measure usually gets disregarded.

At this point, this system evaluates the available information (note: it now has 'complete' information about the characteristics it started with) with respect to making a decision and identifies a gap in the evidence: the DISTANCE characteristic doesn't really contribute any evidence to support either alternative.

```
One of the features doesn't contribute any evidence to the decision.
Is there another characteristic that is significant to the goal,
   COMMUTE? yes
What is its name? miles
Do you have information on the values of the alternatives on
   the feature, Miles?  [y or n] y
What is the value of Charming_Cape for characteristic Miles? 10
What is the value of Castle for characteristic Miles? 20
What is the best value that they can have? 5
Is the difference between Charming_Cape and Castle
   significant on characteristic Miles? yes
Which alternative performs better on miles?
    [CHARMING_CAPE = 1 & CASTLE = 2, 0 = neither]  1
```

The program substituted the new characteristic, MILES, for the insignificant characteristic, DISTANCE. The new state that resulted from the substitution is in some ways less certain than the previous state. The old state permitted a choice because the only significant evidence supported alternative 2. Now however, one characteristic supports alternative 2 and the other supports alternative 1; so there is conflict that must be resolved.

```
Is one of the characteristics, (Miles) and (Roads),
more important than the other, with respect to COMMUTE? n
```

In an effort to resolve the conflict, the program requested importance information. In this case, there is no resolution based on importance.

```
Is there another characteristic that is significant to the goal,
   COMMUTE? y
What is its name? snow-route
```

In the omitted transcript, the program requested explicit information about the characteristic SNOW-ROUTE for the goal COMMUTE. Charming_Cape was on a snow route and the other wasn't. So Sd=1 and Charming_Cape is favored.

```
The characteristic, Snow-Route, is being combined with
   the other characteristics, Miles.
Do they together produce a significant difference?  [y, n or ?] y
⋮
Is one of the characteristics, (Miles Snow-Route) and (Roads),
   more important than the other, with respect to COMMUTE? y
Which is more important?  [0 = (ROADS), 1 = (MILES SNOW-ROUTE)] 1
```

Again following a conservative strategy, the program first requests importance information, as a fallback, and then checks to make sure that there isn't any other evidence.

Since there is no more evidence, the goal, COMMUTE, is evaluated based on the available information and the program suggests a conclusion.

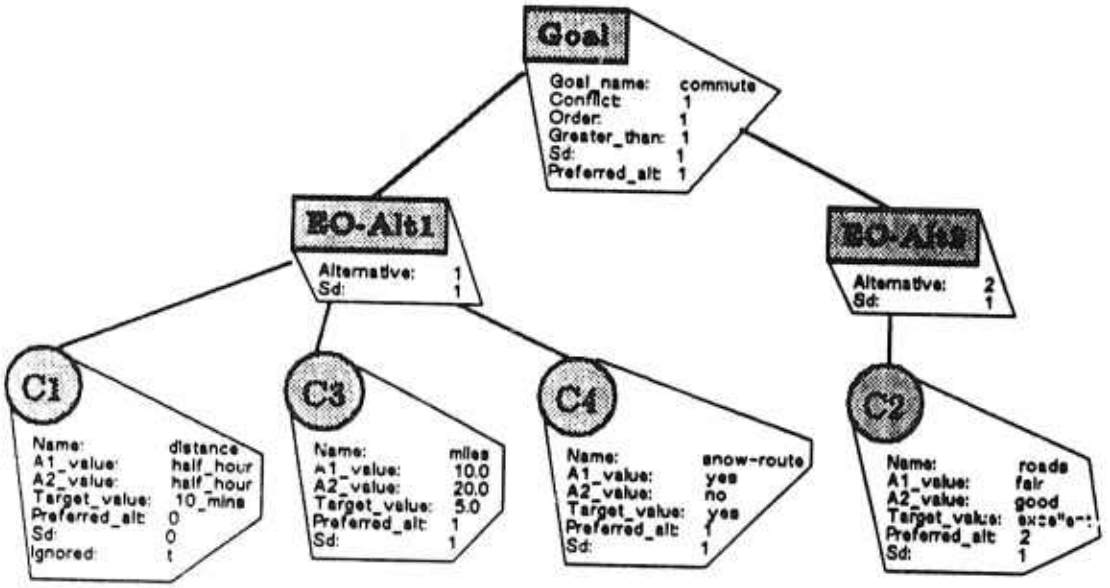
```
Is there another characteristic that is significant to the goal,
   COMMUTE? n
Given the set of characteristics, alternative Charming Cape performs better

   on goal COMMUTE than does CASTLE.
Is that an acceptable conclusion?  [y, n, ?] y
Is the difference between the two alternatives significant on this goal?
y
```

```
To examine all or part of the decision being constructed,
    select one of the following options:
      1  Print the decision tree
      2  Print a goal tree
      3  Explain the current state
      4  Break to lisp temporarily [type (continue) to return when finished]
      5  continue with the program

    which one? 2

Which goal would you like to see?
    1 COMMUTE
  goal: 1
```



```
To examine something else, select one of the following options:
      1  Print the decision tree
      2  Print a goal tree
      3  Explain the current state
      4  Break to lisp temporarily [type (continue) to return when finished]
      5  continue with the program

    which one? 3

The current configuration for the decision is [1, 0, 0, ?, *].

The decision can be described as buying a d. h. jones house.
Alternative 1 is called Charming_Cape. Alternative 2 is called Castle.
All evidence supports the same conclusion. The importance ordering for
the two clusters is unknown currently.
```

With the goal COMMUTE expanded, the other goal, PRIVACY, needs to be worked on. The program expands it in much the same way as COMMUTE. The user provides characteristics: proximity-to-highway and lot-size. The difference between the alternatives on LOT-SIZE is not significant.

```
One of the features doesn't contribute any evidence to the decision.
Is there another characteristic that is significant to the goal,
   PRIVACY? n
Given the set of characteristics, alternative Charming_Cape performs better

   on goal PRIVACY than does CASTLE.
Is that an acceptable conclusion?  [y, n, ?] y

Is the difference between the two alternatives significant on this goal?
n
```

Now two goals have been expanded. Both COMMUTE and PRIVACY support alternative 1. COMMUTE is significant, but PRIVACY isn't.

```
One of the goals doesn't contribute any evidence to the decision.
Is there another goal that is significant to the decision? y
What is its name? pretty_house
```

The user is asked questions about the goal PRETTY_HOUSE. Its characteristics, EXTERIOR and HARDWOOD_FLOORS, support alternative 1. Its characteristic, FIREPLACE, supports alternative 2. The pair of characteristics are more important than the single one.

```
Is there another characteristic that is significant to the goal,
    PRETTY_HOUSE? n
Given the set of characteristics, alternative Charming_Cape performs better

    on goal PRETTY_HOUSE than does CASTLE.
Is that an acceptable conclusion?  [y, n, ?] ?

The best decision seems to be Charming_Cape, alternative 1.
The features which support alternative 1 are preferred by importance ordering.
The current configuration for the decision is [1, 0, 0, ?, *].
The current configuration for the goal PRETTY_HOUSE is [1, 1, 1, 1, 1].

The goal PRETTY_HOUSE has the following values. There doesn't seem to
be a significant difference between the two alternatives on
PRETTY_HOUSE. The currently available evidence is conflicting - there
is some support for each alternative. The evidence in cluster 1 is more
important than that in cluster 2.
The following characteristics support alternative 1:
  exterior (significant), hardwood_floors (significant).

The following characteristics support alternative 2:
  fireplace (significant).

Is Charming_Cape an acceptable conclusion? y
Is the difference between the two alternatives significant on this goal?
y
```

PRETTY_HOUSE gets substituted for PRIVACY. The program requests importance information and is told that PRETTY_HOUSE is more important than COMMUTE.

Another goal, SIZE, is added to the set. It has characteristics, BEDROOMS, BATHS, and GARAGE, supports alternative 2, and is significant. COMMUTE and PRETTY_HOUSE form a more important cluster.

```
Is there another goal that is significant to the decision? n


You can make a decision given the evidence that has been accumulated.
The favored alternative is CHARMING_CAPE.
Is that an acceptable selection? [y, n, ?] ?
```

```
The best decision seems to be CHARMING_CAPE, alternative 1.
The features which support alternative 1 are preferred by importance ordering.
The current configuration for the decision is [1, 1, 1, 1, 0].

The decision can be described as buying a d. h. jones house.
Alternative 1 is called Charming_Cape. Alternative 2 is called Castle.
The currently available evidence is conflicting - there is some support
for each alternative. The evidence in cluster 1 is more important than
that in cluster 2.
Alternative 1 is supported by goals: commute (significant),
  privacy (insignificant), pretty_house (significant).
Alternative 2 is supported by goals:
  size (significant).

Is CHARMING_CAPE an acceptable selection? y
```
"Decision construction completed"

Since there are no more goals, the program recommends an alternative as the decision and gives the user an opportunity to review the evidence supporting it. If it is acceptable, the process ends. Otherwise, the program tries with the user's help to revise the decision.

Because the emphasis was placed on the style of reasoning, rather than the user interface, the interaction is a bit rough. Future versions of the program will include an improved interface with better explanations and some form of sensitivity analysis for allowing the user to consider the repercussions of uncertain judgments. Additionally, as the mechanism is enhanced to include multiple alternatives, the conflict resolution between actions will become correspondingly more sophisticated.

4 How Far Have We Come and What Is Left

The focus of this report is the model of constructive decision making. A decision-maker starts with a two-alternative, two-attribute problem, then acquires information, and perhaps adds attributes and other alternatives, under the guidance of actions associated with decision types. Each decision situation is first classified, then modified by one of the associated actions to make it more tractable. As the information is accumulated, the decision is constructed as a collection of support for one of the alternatives.

CDM contrasts sharply with the more static decision theoretic models. A summary of the differences appears in Figure 4.6[3]. The goal of decision theory is to find optimal

[3]This table was produced with help from Tammy Tengs, a member of the department of Operations Research at UMass.

	Decision Theory	Decision Typology
Goal	optimizing	satisficing
Algorithm	combines evidence	gathers evidence
Evaluation	static	dynamic
# of alternatives	many	2
Comparison scales	single	multiple
Informational burden on user	reductionistic	holistic
consistency	required	ignored
Ignorance of attribute values	assume some distribution	disregarded or deferred
Ignorance of attribute importance	assume equality	disregarded or deferred
Requirements on attributes	measurable sufficient minimal non-overlapping	none
utility theory	important & explicit	not explicit
numeric/symbolic	probabilities	reasons
cost of evidence	included	included

Figure 4.6: Comparison of Decision Theory and Decision Typology

solutions. In CDM, satisficing is preferred so that the search space will be manageable. CDM is dynamic because the algorithm gathers evidence as the decision evolves. Evidence is not combined on a single scale, but rather is compared as collections of support. CDM manages uncertainty and ignorance by gathering available information as supportive evidence at the most opportune time.

The method circumvents mapping attributes to a uniform scale by abstracting their difference according to whether or not the attributes provide support to a decision. However, this advantage comes at a cost. As implemented in the decision support system, the decision typology places the burden of assessment on the user. In the typology, the user is required to make qualitative assessments of combinations of support as the attributes are included in the evolving decision.

Yet, integrating the assessments into the construction process is integral to making dynamic control decisions in AI programs. CDM provides this integration and a satisficing strategy that permits faster decisions if the domain requires quicker response.

Constructive decision making has not yet been included as a decision making component of an AI program. In fact, some issues must first be resolved. Because the burden of preference combination is placed on the user, the mechanics of deriving dimensions and producing preference judgments must be worked out. More importantly, the conditions and mechanisms for adding new alternatives must be specified as they were for new attributes.

Constructive decision making is the core of a decision support system, CDM. Hwang and Yoon in their book on multi-attribute decision making (Hwang and Yoon 1981) expressed some frustration in the future/current work section of their book about the lack of easy to learn, useful methods for decision support. Sage and Rouse (1986) in reporting the results of a workshop on aiding the human decision maker emphasize the need for research into systems that help structure decision problems and address dynamic decision situations. CDM addresses these concerns through an iterative method that more closely emulates human decision making styles than standard decision techniques, and qualitative assessments that are more comfortable for people to provide.

CDM is not intended to produce optimal solutions to complex decision problems given complete information, but rather to explore methodologies for structuring decision problems, performing symbolic comparisons, reasoning about uncertain decisions, and automating dynamic decision making.

Chapter 5

Task-level Architectures and Knowledge Engineering

A *knowledge system architecture* is a level of description of knowledge systems that specializes general AI implementation techniques to suit a class of problem solving tasks. This report presents three complementary views of the architecture level, and analyzes their implications for the design of knowledge engineering tools. The analysis is illustrated with an architecture for systems that reason under uncertainty, and with a hierarchy of knowledge engineering tools to support system development and knowledge acquisition at the architecture level.

1 Introduction

This report is about tools for knowledge engineering at the *architecture level.* A knowledge system architecture specializes common AI problem-solving techniques to a particular class of tasks. Architectures provide descriptions of particular kinds of problem solving (e.g., *diagnosis* or *configuration*) at a conceptual level that is above the implementation, thus making clear which aspects of a class of problems are intrinsic to the problem and which are artifacts of the implementation. Architectures are partial designs in which some decisions are made in advance to support particular task characteristics. For example, many medical diagnosis systems first interpret data bottom-up to find "triggered" disease hypotheses, then set top-down goals of acquiring evidence pro and con the triggered hypotheses. This "trigger/acquire evidence" cycle is an intrinsic part of any architecture for the class of medical diagnosis tasks, though it might be implemented in a wide variety of ways.

Architecture-level tools for knowledge engineers can improve the productivity of system development and knowledge acquisition because:

- By supporting the abstraction of representational and computational primitives

at the architecture level, they permit the knowledge engineer and expert to cooperatively develop systems using a shared language of architecture constructs, rather than in terms of the underlying implementation.[1]

- They can incorporate knowledge about the architecture to facilitate system development and knowledge acquisition (e.g., by enforcing constraints on the types and values of elements in the knowledge base).

The idea of an architecture level underlies recent work on knowledge systems.[2] Chandrasekaran and his colleagues have identified a number of "generic tasks" such as hierarchical diagnosis and routine design, and have developed task-specific representation languages and control strategies for them (Chandrasekaran 1986, Bylander and Mittal, 1986, Brown and Chandrasakeran, 1985). McDermott and colleagues have produced several knowledge systems using architectures that integrate knowledge acquisition tools with the problem solving methods (Kahn et al., 1984, Eshelman and McDermott, 1986, Marcus, 1987, Kahn et al., 1987). Clancey has described in detail the heuristic classification method embodied in the HERACLES architecture (Clancey, 1986). Newell (Newell, 1982) anticipated much of this work in his AAAI President's Address on the *Knowledge Level*, where he distinguished the knowledge of an intelligent agent, which is used to model its behavior, from the knowledge representation that describes how the knowledge is encoded in a symbol system.

This report presents an analysis of the role of knowledge engineering tools at the architecture level. We describe three complementary views of what is meant by the architecture level, and illustrate them in the context of MU, an architecture for systems that manage uncertainty by deciding how to act. We show how the architecture-level analysis leads to a hierarchical organization of knowledge engineering tools to support software development and knowledge acquisition for MU systems. We conclude with some advantages of this approach to knowledge engineering.

2 Three views of the architecture level

Architectures can be viewed from three perspectives, each which suggests roles for architecture-level tools. First, the *functional* view presents an architecture as an application of general AI techniques to suit a particular style of problem solving. One might

[1]Data abstraction and related methodologies such as object-oriented programming are well established software engineering techniques for reducing the complexity of large programs by hiding implementation details (Abelson 1985). For knowledge systems, the architecture is a particularly useful level of abstraction, and tools to support it reduce the inherent complexity of large knowledge-based programs by separating the representational and computational needs of the problem solving task from implementation decisions.

[2]The architecture level was a major focus of the AAAI Workshop on High-level Tools in October, 1986. An earlier version of this paper was presented there.

say that, functionally, the blackboard architecture is well-suited to problems with noisy data and multiple sources of evidence (HEARSAY 1980, Nii 1986). There are architectures for simple classification (e.g., traversing decision trees), heuristic classification (e.g., HERACLES (Clancey, 1986); CSRL (Bylander and Mittal, 1986)), constructing configurations (e.g., SALT (Marcus, 1987); COAST (Bennett, 1986)), and design (e.g., DSPL (Brown and Chandrasakeran, 1985)); DOMINIC (Howe et al., 1986)).

The second perspective is *structural*: an architecture is a partial design that includes specifications of knowledge representation formalisms, inference mechanisms, and control strategies. Many of these structural components are available from commercially available AI programming environments. Architectures, however, are not arbitrary combinations of these components, but "good" combinations designed by the knowledge engineer for particular tasks.

A third view of an architecture is that it defines a *virtual machine*. The architecture provides a language that describes the behavior of a system in terms natural for the knowledge engineer and expert. For example, most medical diagnosis systems provide some kind of support for *triggering* – making particular hypotheses "active" when certain events (typically input data) occur. To the expert, triggering might correspond to "bringing a diagnosis to mind." A programmer can produce the effect of triggering using implementation-level primitives (e.g., giving triggered diseases high certainty factors or agenda priorities). But terms such as "trigger" — not their implementation — are the medium of knowledge engineering. Such *task-level* terms promote explanation (Swartout, 1983) and knowledge acquisition[3] (Gruber and Cohen, 1987). Knowledge engineers, experts, and users can all understand triggering without thinking about how it is implemented. A virtual machine that executes "triggering" is easier to program.

In summary, the functional view of an architecture emphasizes the behavior of programs that instantiate it. The structural view emphasizes knowledge representations, inference methods, and other components of the architecture. A virtual machine integrates these views: it is an abstract device designed to meet the functional needs of a class of problem solving tasks. The next section discusses how the interactions of these views result in an organization of knowledge engineering tools.

3 Tools for the MU Architecture

In this section we present an architecture for systems that reason under uncertainty, called MU (Cohen et al., 1987b), with the aim of illustrating how the three views of architectures influence the design of knowledge engineering tools. MU grew out of experience with MUM (Managing Uncertainty in Medicine), a system for planning a series of diagnostic questions, tests, and treatments for diseases manifesting chest and

[3]Without task-level terms, the (non-programmer) expert is effectively barred from working directly with the knowledge base.

abdominal pain (Cohen et al., 1987a). The primary aim of MUM is to decide how to act when data are insufficient for diagnosis and treatment. Like a physician, MUM reasons about tradeoffs between the costs of evidence, the marginal utility of potential data given what is already known, the effects of treatments and the evidence they provide, and so on. MU is an architecture for building systems like MUM that reason about uncertain situations in deciding how to act.

Viewed from a *functional* perspective, MU's task is *managing uncertainty* by taking appropriate actions. *Structurally,* MU has a large long-term memory of hypotheses and their supporting evidence and intermediate conclusions, a working memory of developing hypotheses, inference mechanisms for propagating the effects of evidence in working memory, and control strategies. Viewed as a *virtual machine,* MU supports knowledge engineering in terms that make sense for diagnostic tasks, such as *hypothesis* and *potential-evidence.* These terms are instantiated for specific domains by terms such as *disease*, or further instantiated as specific diseases such as *angina.*

The interactions of these views of the MU architecture are apparent in the design of knowledge engineering tools. Figure 5.1 shows a hierarchy of tools that supports development of systems in MU. The foundation is a commercially-available AI programming environment that includes implementation primitives such as rules and frames, and basic AI programming techniques such as pattern-matching rule interpreters and message-passing. The first layer in Figure 5.1 is a *structural* description of the implementation of MU. It is not a design for an architecture, because no *functional* description has been given or is implied by this collection of implementation primitives and AI programming techniques, which could be instantiated to provide a wide range of behaviors.

The functional view of an architecture constrains how implementation-level primitives and techniques are *specialized* for a particular kind of problem-solving. The functional requirements of MU are that it should represent inferential relations between data, intermediate conclusions, and hypotheses. It should maintain measures of belief in all these objects, decide focus of attention (i.e., which objects to seek evidence for), and decide which evidence to seek. At the second level of Figure 5.1, the frames and slots of the first level are specialized as *hypotheses* and *inferential relations*, respectively. Rules are used to implement *combining functions* for evidence pro and con hypotheses. Some properties of hypotheses – a subset of their slot values – are used as *control parameters*, which help determine focus of attention. Similarly, value propagation functions are implemented via the demons and message passing, and so on. Thus, the functional view of the MU architecture tells the architecture designer how to specialize low-level implementation primitives and techniques to achieve a *virtual machine*, or shell, for a particular class of tasks.

An architecture is designed not for a specific task like diagnosing chest pain, but for a class of tasks such as diagnostic reasoning. Thus, the knowledge engineer and expert must *instantiate* architecture-level primitives for a particular application just as the architecture designer needed to specialize implementation-level primitives. Figure 5.2

Tool Level	Objects in User's View	Software Support
Knowledge Acquisition Interface	*Application-specific Terms* diseases, intermediate diagnoses, questions, clinical tests, triggering symptoms for diseases, confirming test results, criticality of diseases, relative costs of tests, treatments, efficacy of treatment	*(Meta-)Knowledge-based Utilities* language-specific editors and form-filling interfaces, inferential consistency analyzer, graphical display for objects and relations
Virtual Machine (shell)	*Task-level Constructs* hypotheses, intermediate conclusions, inferential relations, data descriptions, combining functions, control parameters, control rules, preference rankings among actions	*Task-specific Reasoning Mechanisms* value propagation functions, predicates on the state of the inference net, rule-based planner, decision-making support
AI Toolbox (KEE)	*Implementation Primitives* frames and slots, rules, pattern matching language, Lisp objects and functions, windows and graphic objects	*AI Programming Techniques* knowledge base bookkeeping, rule interpreter, knowledge base bookkeeping, inheritance mechanisms, assumption maintenance, demon invocation and message passing, window system, network grapher

Figure 5.1: A hierarchy of knowledge engineering tools to support the MU architecture.

is a structural view of MUM – a chest pain specialist – engineered in the MU architecture. Hypotheses are instantiated as *diseases* such as *classic angina*; intermediate conclusions are instantiated as *clusters* such as *exercise-induced-pain*; inferential relations such as *potential evidence* are instantiated by specific links between evidence and conclusion, such as *EKG results* and *classic angina*.

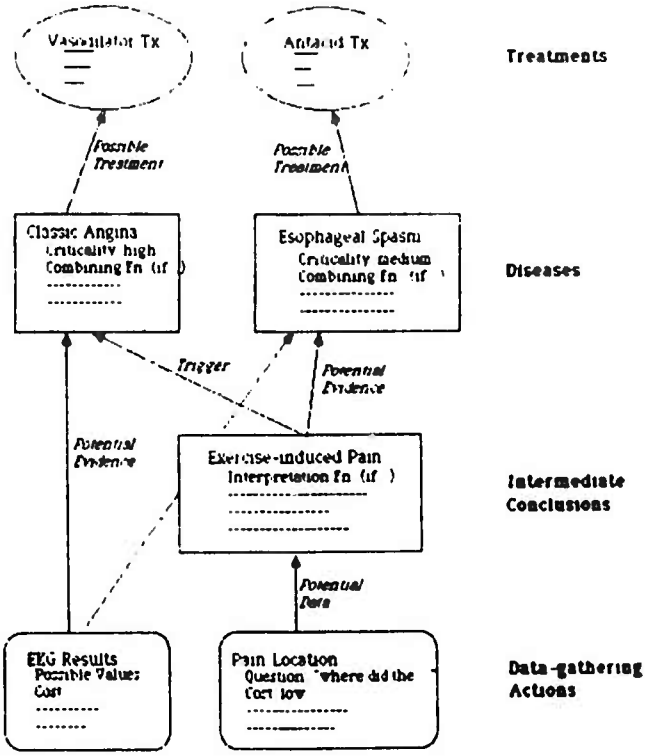


Figure 5.2: Fragment of the inference network for MUM

Once the knowledge engineer has decided to instantiate hypotheses as diseases, he or she can build a knowledge-acquisition interface to help elicit knowledge in the terms of the architecture. Meta-knowledge about the terms is provided by the knowledge engineer while designing the shell, and is used by the knowledge acquisition interface to help the user build a syntactically valid and semantically consistent knowledge base. We currently have form-filling editors for all objects in Figure 5.2, a graphics interface for acquiring some continuous combining functions, and rudimentary consistency-checking abilities; other tools, especially for acquiring control knowledge, are in progress.

4 Conclusions

Architecture-level knowledge engineering tools have several advantages:

- One can capitalize on vertical integration of implementation-level tools at the architecture level. For example, a general-purpose frame editor and network grapher that is provided at the implementation level (such as the KREME interface (Abrett and Burstein, 1987)) can be customized as a knowledge acquisition interface for editing architecture-level constructs such as *hypotheses* and *diseases*. This is possible because the architecture-level objects are specializations of implementation-level objects.

- Software development is facilitated because architecture-level constructs — the primitive objects of the virtual machine — are represented declaratively. For example, once the *trigger* relation has been designed, one need not worry about several members of a software project trying to achieve its functionality by different implementations.[4]

- Declarative architecture-level constructs also facilitate knowledge acquisition because meta-knowledge (Davis 1984) can be attached to objects to check for consistency, provide help, generate explanations, and so on. For example, a form-filling interface specialized for acquiring an instance of a disease can use a declarative description of the properties of diseases, such as the kinds of relations they have with data, to offer a menu of documented choices (Gruber and Cohen, 1987).

- Building a virtual machine at the architecture level and then a knowledge acquisition interface on top of the virtual machine defines the roles of the knowledge engineer and expert. The knowledge engineer *designs* an architecture by specializing general-purpose implementation-level tools to operationalize the constructs suited for the problem solving task, whereas the expert *instantiates* architecture-level constructs for the application domain. Virtual machine tools (shell support) assist the knowledge engineer in customizing an architecture for a particular application, and knowledge acquisition tools help the expert build, refine, and debug the knowledge base.

5 Discussion

The hierarchy of tools discussed here reflects a power/generality tradeoff. Constructs at the implementation level are general (e.g., production systems can be configured for many kinds of problem solving) but from the standpoint of knowledge engineering they are weak. To say an object is a *disease hypothesis* is to imply much more knowledge about it than to say it is a *frame*, even though the implementation

[4]The EES project (EES 1985) aims at making *every* implementation decision explicitly recorded in a language which allows a program writer to actually generate the code.

of the disease hypothesis may be no more than a frame. This added knowledge constrains the internal structure of the disease frame (e.g., values and types of slots, or the kinds of messages it can handle, etc.), constrains its relationships with other frames, and so on. Since these constraints facilitate knowledge engineering, architecture-level objects like disease frames are at the "power" end of the power/generality spectrum. Implementation-level objects, lacking constraints, are more general but correspondingly less powerful from the standpoint of knowledge engineering.

Thus, when one builds an expert system for a task, the utility of an architecture level analysis depends entirely on how much one *knows* about the task. The knowledge system architecture embodies knowledge about a class of problem solving tasks – it is a virtual machine for that class – and as such facilitates system development and knowledge acquisition for problem solvers of that class. The power/generality tradeoff tells us that we can ameliorate the knowledge acquisition bottleneck for restricted classes of tasks by designing architectures and building integrated "power tools" at the architecture level.

The problem of knowledge acquisition is viewed in terms of the incongruity between the representational formalisms provided by an implementation (e.g., production rules) and the formulation of problem solving knowledge by experts. The thesis of this report is that knowledge systems can be designed to facilitate knowledge acquisition by reducing representational mismatch. Principles of *design for acquisition* are presented and applied in the design of an architecture for a medical expert system called MUM. It is shown how the design of MUM makes it possible to acquire two kinds of knowledge that are traditionally difficult to acquire from experts: knowledge about evidential combination and knowledge about control. Practical implications for building knowledge acquisition tools are discussed.

6 Design for Acquisition

Knowledge acquisition is the process of gathering knowledge about a domain, usually from an expert, and transforming it to be executed in a computer program. It is a part of the knowledge engineering process, which includes defining a problem, designing an architecture, building a knowledge base, and testing and refining the program. Knowledge acquisition is regarded as the bottleneck in this process. Our thesis is that the design of a knowledge system should anticipate the acquisition process. By analogy with "design for testability," in which digital hardware is designed to be easily tested (Bennetts, 1984), our aim is *design for acquisition*: designing knowledge systems to facilitate knowledge acquisition.

The first advance on the knowledge acquisition problem was the invention of *general architectures:* knowledge representation techniques and accompanying interpreters that allow the programmer to encode domain knowledge in a knowledge base separate

from the algorithm that interprets it. The EMYCIN architecture is paradigmatic (van Melle, 1979; see also Buchanan and Shortliffe, 1984). Its essential architectural features are a rule formalism with conjunctive premises, certainty factors, and an exhaustive backward-chaining control strategy.

With the general architectures came tools to help the knowledge engineer and expert transform knowledge into the available formalisms. Experts were insulated from the Lisp implementation by rule editors and pseudo-natural language interfaces (Shortliffe, 1976). In TEIRESIAS, Davis (1976) demonstrated that a knowledge acquisition program can use knowledge about the architecture, such as the structure of rules and the effect of backward chaining, to help users refine and debug the knowledge base. With ROGET, Bennett (1985) showed that information about the kinds of domain knowledge likely to useful for a task could be used by a system to help acquire the initial "conceptual structure" of a domain.

Recently, knowledge systems research has emphasized the power of less general, more task-specific architectures (Chandrasekaran, 1986; Clancey, 1985; McDermott, 1983). Many systems share common problem solving methods, despite differences in implementation. When the task can be characterized at a level independent of the implementation, an architecture can be designed to capture the task-specific problem solving knowledge. For example, the HERACLES architecture (Clancey, 1986) is designed to do *heuristic classification*, a common task for knowledge systems.

Knowledge acquisition tools for task-specific architectures can apply knowledge about the kind of problem that the task addresses and the problem solving methods it provides. For example, ETS (Boose, 1984) is a method of acquiring knowledge for hierarchical classification tasks. It applies a psychological theory of how to elicit classification hierarchies from people. SALT (Marcus, McDermott, and Wang, 1985) assists in knowledge acquisition for iterative design tasks such as configuration. The architecture for SALT identifies three kinds of domain knowledge used by its problem solving strategy, and SALT uses knowledge about their form and purpose to focus and constrain the knowledge acquisition dialog. In both cases, the knowledge acquisition method is driven by demands of the task (e.g., classification or configuration) rather than the implementation formalisms (e.g., rules).

Both ETS and SALT acquire knowledge for well-characterized problem solving methods, with corresponding architectures. However, the space of methods (and even tasks) for knowledge-based systems has surely not been fully explored. For those problems without suitable task-specific methods, knowledge systems and tools to build them must be *designed*. Design means choosing knowledge representations and control strategies that can bring expert knowledge to bear on the problem. Careful attention to the design of a problem solving architecture can make knowledge acquisition easier both for knowledge engineers and automatic knowledge acquisition tools.

This report presents principles for designing knowledge systems to facilitate knowledge acquisition. In Section 7, we introduce three general principles of design for

acquisition. In Section 8, we show how these principles have been applied in the design of an architecture for managing uncertainty in medicine. The architecture makes it possible to acquire two kinds of knowledge that are traditionally difficult to acquire: knowledge about evidential combination and knowledge about control. In Section 9 we show how the principles of design for acquisition imply a hierarchical organization of tools for implementing knowledge system architectures, emphasizing the integration of knowledge acquisition support.

7 Principles of Design for Acquisition

This section presents three principles that should be considered during the design of a knowledge system architecture. They may be familiar to knowledge engineers as heuristics for knowledge representation. We emphasize the goal of making it easy for *experts* to express their knowledge.

Principle 1:

- *Design task-level representational primitives to capture important domain concepts defined by the expert.*

This principle prescribes that the knowledge engineer provide a language of task-level terms. It addresses a fundamental obstacle to knowledge acquisition, the *representational mismatch* between the way that an expert formulates domain knowledge and the way the knowledge is represented in an implementation (Buchanan, Barstow, Bechtel, Bennett, Clancey, Kulikowski, Mitchell, and Waterman, 1983). Representational mismatch typically occurs when the knowledge engineer imposes implementation-level primitives on the expert. For example, knowledge acquisition in a strictly rule-based architecture is ultimately *rule* acquisition, and if it is difficult for an expert to express problem solving expertise as rules, then it is hard to acquire the knowledge. The problem is that rules are implementation-level primitives.

An example of a *task-level* primitive is the notion of a "trigger" — a special relation between data and hypotheses such that when the data are found, a hypothesis is immediately activated. Trigger is natural construct for diagnosticians. For the cardiologist, a 45 year old man complaining of chest pain with exercise "brings to mind" the hypothesis of angina. If it is a representational primitive for the system, then acquiring triggering relations from the expert is straightforward. If instead one must achieve the effect of a trigger by, say, "tuning" the certainty factors of rules or the weights on links, then it will be difficult to acquire, explain, and modify this knowledge.

Principle 2:

Design explicit, declarative representational primitives.

From the standpoint of knowledge acquisition, declarative knowledge representations are preferable to procedures. The meaning of declarative representations can be "read" directly, whereas the meaning of procedures can only be had by executing the procedure or simulating its execution. For experts to understand procedural representations of their knowledge they must first understand the algorithm that interprets them. Even when knowledge seems naturally represented with procedures (e.g., control knowledge), formulating it declaratively can facilitate acquisition, explanation, and maintenance. Furthermore, the requirement of explicitness means that when a new primitive is needed to express some domain concept or expert strategy, its purpose and operational definition must be made public; this is important when multiple experts and programmers work on a system.[5]

In Section 8.3 we show how designing declarative primitives for control knowledge allows one to represent expertise in deciding what to do next under conditions of uncertainty. By making the knowledge explicit and declarative, the expert can examine the assumptions underlying his or her control decisions. In Section 9 we show how representing task-level terms declaratively allows the use of conventional "form-filling" user interface technology in knowledge acquisition tools.

Principle 3:

Design representations at the same level of generalization as the expert's knowledge.

This principle can be summarized with two caveats:

- Don't force experts to generalize except when necessary.
- Don't ask experts to specify information not available to them.

Generalization is one of the dimensions of representational mismatch, the distance between the expert's formulation and the implementation.[6] A representation and its referent in the world are at different levels of generalization if there are distinctions in the world that the representation fails to capture or the representation makes artificial distinctions. An example of overgeneralization is forcing a large range of values into a small set of categories. The expert interpreting blood pressure considers the full range of systolic/diastolic ratios, while the knowledge engineer may want to categorize it as `high`, `normal`, or `low`, to make it easier to implement. Conversely, the knowledge engineer may ask the expert to specify more knowledge than he or she has, again to suit the implementation. For example, the expert may be asked to supply degrees of belief with far more precision that is justified by his or her knowledge.

[5] Neches, Swartout, and Moore (1985) emphasize the advantages of this principle for explainability and maintainability, and Clancey (1983) argues for the explicit representation of control knowledge to facilitate explanation, knowledge engineering, and tutoring. Our point here is that "good engineering" is also good for knowledge acquisition.

[6] Another dimension is *operationalization*, converting advice to procedures (Mostow, 1983).

8 A Case Study of Design for Acquisition

In this section we illustrate the principles of design for acquisition in the context of a medical expert system. We show how the design of the system facilitates acquisition of two kinds of knowledge that are traditionally hard to acquire: knowledge about how to combine evidence and knowledge about how to control the order of actions. The ability to capture this expertise gives the system a unique ability to *manage uncertainty* by selecting or planning actions that will minimize uncertainty or its effects.

8.1 Task domain: Prospective Diagnosis

MUM is an knowledge system that Manages Uncertainty in Medicine, currently in the domains of chest and abdominal pain. (See Cohen, Day, Delisio, Greenberg, Kjeldsen, Suthers, and Berman, 1987, for details.) Physicians make a distinction between *retrospective diagnosis*, in which all the evidence is known in advance and the goal is to make the correct diagnosis, and *prospective diagnosis*, which emphasizes the proper management of the patient through the *workup*, a diagnostic sequence of questions and tests. In prospective diagnosis, uncertainty about the patient's condition is managed by gathering evidence in the best order (e.g., to maximize diagnostic information and therapeutic effectiveness and to minimize cost and discomfort). MUM's task is prospective diagnosis; it uses expert knowledge about evidence and control to generate an intelligent workup for a patient.

The knowledge acquisition task for MUM includes not only eliciting *heuristic associations* (Clancey, 1985) between evidence and diseases ("What are the symptoms of angina?"), but also *combining knowledge* ("What is the effect of risk factors like smoking on the hypothesis of angina when there has only been one episode of pain?"), and *control knowledge* ("Under what conditions should an angiogram be given?").[7] The expert for MUM has a wealth of combining and control knowledge, central to his expertise as a physician. This knowledge is difficult to represent, and thus acquire, in current architectures. We designed MUM in accordance with the principles discussed above to make it easy to acquire combining and control knowledge.

8.2 Design for acquisition of combining knowledge

Combining knowledge specifies how belief in several pieces of evidence is combined to support a single conclusion. Remarkably, knowledge engineers rarely ask experts how they combine evidence. Instead, fixed, global numeric functions that compute degrees of belief are built into the architecture (Duda, Hart, and Nilsson, 1976; Shafer, 1976; Shortliffe and Buchanan, 1975; Zadeh, 1975). Although the numeric representations

[7] An angiogram is an expensive, invasive test for coronary artery blockage, usually given only after other tests show positive results.

and functions are a convenient implementation formalism, they make it surprisingly difficult for experts to express their knowledge about how they manage uncertainty (Cohen and Gruber, 1985; Szolovits and Pauker, 1978).

MUM's design does three things to facilitate the acquisition of combining knowledge. First, it replaces the real-valued numeric representation of uncertainty with symbolic *states of belief* that are meaningful in domain terms. Second, it provides an explicit representation for *clusters of evidence*, to encapsulate diagnostically significant subsets of evidence. Third, it replaces the global numeric function with *local combining functions*, specified by the expert, for each cluster of evidence.

MUM represents belief as ordinal values that characterize the expert's evaluation of evidential support. Seven states of belief are defined by the expert: `confirmed`, `disconfirmed`, `supported`, `detracted`, `strongly-supported`, `strongly-detracted`, and unknown. They are primitives at the task level; each has diagnostic or therapeutic significance.

MUM represents combinations of evidence with *clusters*, frames that represent diagnostically significant groupings of evidence. With respect to evidential support, diseases are clusters. Clusters also represent intermediate results, such as common groupings of clinical findings and definitional data abstractions (Clancey, 1985). For example, the cluster `chest-pain-when-eating` illustrated in Figure 5.3 describes a situation in which the chief complaint of a patient is pain or pressure in the chest that begins after eating. This cluster *triggers* the disease classic-esophageal-spasm. `crescendo-pain-long-duration` in Figure 5.3 represents the situation where the pain has been increasing in intensity for more than ten minutes. The cluster discriminates between angina and esophageal spasm: pain from the former usually lasts less than ten minutes.

MUM represents evidential combination with *local combining functions*, symbolic functions mapping states of belief in evidence to states of belief in a conclusion. Each cluster has its own combining function, and there are no global combining functions. Combining functions are acquired from the expert, usually as a set of rules, but they can also be acquired in tabular or graphical form. The combining function for the first cluster in Figure 5.3 is a simple example; if an episode of chest pain (which may also be described as pressure) is incited by eating then this is a `confirmed` case of `chest-pain-when-eating`. No other combination of states of belief in evidence has any affect on belief in that cluster. Diseases, also represented as clusters, typically have more complex combining functions. For example, the frame for `classic-esophageal-spasm`, with the set of rules that define its combining function, is shown in Figure 5.4.

The representation of combining knowledge in MUM is unconventional, but not novel. (Similar designs are used in PIP (Pauker, Gorry, Kassirer, and Schwartz, 1976), MDX (Chandrasakeran, Mittal, and Smith, 1982), and the "criteria tables" of (Kingsland and Lindberg, 1986).) It was, however, designed to facilitate knowledge acquisition, in accordance with the principles of design for acquisition. First, applying Princi-

Cluster: `chest-pain-when-eating`
Combining-function:
IF (and (or (confirmed episode-chief-complaint=pain)
(confirmed episode-chief-complaint=pressure))
(confirmed episode-chief-complaint-location=chest)
(confirmed episode-incited-by-eating))
THEN confirmed

Cluster: `crescendo-pain-long-duration`
Combining-function:
IF (and (or (confirmed episode-chief-complaint=pain)
(confirmed episode-chief-complaint=pressure))
(confirmed episode-chief-complaint-frequency=crescendo)
(confirmed episode-chief-complaint-duration>10minutes))
THEN confirmed

Figure 5.3: Two clusters for diagnosing chest pain

Clusters represent diagnostically significant combinations of evidence. They might play a part in a diagnostic scenario like this: A patient reports an episode of chest pain incited by eating (`chest-pain-when-eating`); this combination of findings is relevant to many diagnoses. (For instance, it *triggers* `classic-esophageal-spasm`, shown in Figure 5.4.) The physician then asks about the duration and time course of the pain. If the report matches the situation characterized here as `crescendo-pain-long-duration`, the cluster is confirmed. This cluster of symptom descriptions is useful in differential diagnosis; for instance, it *supports* `classic-esophageal-spasm` and *detracts* `classic-angina`). In these examples, the combining functions specify necessary and sufficient conditions for clusters to be confirmed; no other belief state (such as *supported*) is relevant.

Cluster: `classic-esophageal-spasm`
Isa: disease
Triggered-by: (confirmed chest-pain-when-eating)
Combining-function:
IF (or (confirmed barium-swallow=spasm)
(confirmed manometrics))
THEN confirmed
IF (or (confirmed vasodilator-tx)
(confirmed nitroglycerin-tx))
THEN strongly-supported
IF (confirmed crescendo-pain-long-duration)
THEN supported
IF (disconfirmed nitroglycerin-tx))
THEN detracted
IF (confirmed chest-pain-short-duration))
THEN detracted
IF (disconfirmed barium-swallow=spasm)
THEN disconfirmed

Figure 5.4: Part of a Disease Frame for Classic Esophageal Spasm
The evidential combining function for this disease is made up of rules; each *IF* part specifies conditions on the state of belief in clusters, and each *THEN* part asserts a state of belief for the disease. "tx" means trial therapy; for example, nitroglycerin-tx is confirmed if pain goes away when the patient takes a nitroglycerin tablet. Manometrics and barium-swallow are tests; barium-swallow=spasm is a cluster that is confirmed when the barium-swallow shows a spasm. The triggering function has the same syntax as the left hand sides of rules in the combining function. In this example, when the cluster chest-pain-when-eating (Figure 5.3) is confirmed, the disease classic-esophageal-spasm is triggered. These combining and triggering functions were elicited by a knowledge engineer working with a physician, in the context of actual cases. From inspecting the combining function, a planner can infer that the tests (manometrics and barium-swallow) are most diagnostic, since they can confirm and disconfirm the diagnosis of this disease. In the frames representing these tests, however, one will find that they are invasive and therefore costly – to be avoided. Slightly less diagnostic information (strongly-supported, detracted) can be obtained from trial therapies, and even less (supported, detracted) from reports of episodes of pain given by the patient.

ple 1, the ordinal states of belief are chosen to be sufficient to characterize diagnostically significant situations, and nothing more. Since the expert defines these terms, there is no problem of "getting numbers from experts." The expert can state categorically the implications of a subset of findings, instead of relying on the system to calculate a partial match to a set of possible findings, as in INTERNIST (Pople, 1977). Second, symbolic combining functions are explicit, declarative representations of decisions about evidential support, whereas the belief in a conclusion given belief in evidence is only implicit in global, numeric combining functions. Adhering to Principle 2 in this case means representing evidential *judgments*, rather than representing degrees of belief and *computing* the result. Third, symbolic, local combining functions represent specific combinations; only a subset of possible evidence is considered for each cluster, and only some of the belief states in each constituent piece of evidence are specified. This contrasts with the situation where *no* local combining function is specified, and *every* possible combination of belief is possible. In accordance with Principle 3, the expert is not asked for information (e.g., probabilities) that can be used to make distinctions (e.g., in levels of belief) that he or she does not endorse.

We have found that having to specify the combining knowledge explicitly and locally makes knowledge acquisition more efficient when maintenance and knowledge base refinement are considered. Combinatorial problems are avoided because the space of combinations is very sparse; not *every* combination of belief in every piece of evidence is relevant in the chest pain domain. This holds advantages for knowledge base refinement and testing. First, every combination of evidence is justified. Second, when test cases are found for which combining knowledge is inadequate, the omission is easily localized to the cluster where the combining function is underspecified. If combining functions produce conflicting belief states for the same cluster, it indicates a case that the expert had not considered, an error of omission. Thus the design helps experts, knowledge engineers, and knowledge acquisition tools address the credit assignment problem.

8.3 Design for acquisition of control knowledge

A major part of expertise in *prospective* medical diagnosis is the ability to gather data in the proper order, omitting unnecessary tests, asking only those questions that pertain to relevant hypotheses, and prescribing preliminary or exploratory treatment before all of the manifestations of a disease are present.[8] This is *control knowledge* about what to do, rather than what to believe. Traditionally, domain knowledge is acquired without troubling the expert to think about control. Simple control strategies such as forward chaining are implicit in the interpreter, separated from the domain knowledge base, and selected by the knowledge engineer. When these weak methods

[8]Prospective diagnosis is concerned with gathering evidence for diagnosis and treatment, and is fundamentally different from retrospective diagnosis which concentrates on the classification of data already available.

are inadequate, the knowledge engineer coerces the interpreter to do something more complicated, perhaps by ordering rules or having rules communicate via control flags. Other techniques for specifying control, such as procedural attachment in frame-based systems, are, again, designed and implemented by knowledge engineers largely without consulting experts. But experts have useful domain-specific knowledge about *how* to solve problems that should be acquired.

The problem we faced in MUM was how to represent control knowledge so we could acquire it from the expert. The solution is to ask the expert for the parameters of a domain that affect control decisions, and then ask him to formulate control knowledge in terms of these control parameters. For example, some diseases are more *dangerous* than others; some clinical tests are very *costly*; and some evidence is more *diagnostic*. Control knowledge is easier to acquire in these *task-level* terms, in contrast to *implementation-level* parameters, such as the priorities of tasks on an agenda, or the order of clauses in a rule. Since task-level control parameters are declarative they can be reasoned about by a knowledge-based system, and more to the point, they can be acquired.

Control parameters are a vocabulary for describing situations in which the expert KNOWS what to do. *Control rules* (Davis, 1976), acquired from the expert in terms of control parameters, represent the decision points in diagnosis. Given the evidence that has already been acquired, and the hypotheses it suggests, the diagnostician selects some action, typically to gather evidence for a suspected hypothesis, sometimes by prescribing trial therapy. MUM was designed to represent this decision-making process, so that the expert could specify how it should proceed.

Some control rules specify preferences among alternative actions. For example,

Control rule: prefer-cheap-to-confirming
Conditions: $action_1$ is potentially-confirming, and
$action_2$ is potentially-supporting, and
$action_1$ costs more than $action_2$
Strategy: prefer $action_2$

The effect of this rule is to cause the system to favor cheaper actions and sacrifice a little support.[9] Other control rules specify focusing strategy:

Control rule: focus-on-dangerous-supported-hypos
Conditions: $hypothesis_1$ is supported, and
$hypothesis_2$ is supported, and
$hypothesis_1$ is more dangerous than $hypothesis_2$
Strategy: focus on $hypothesis_1$

[9]Features of evidence like *potentially-confirming* are derived from descriptions of the actions and the clusters for which they serve as evidence. An action (e.g., running a test) is *potentially-confirming* if it can result in evidence that contributes to a *confirmed* state of belief in a cluster (e.g., a disease).

This rule directs the attention of the system to the most dangerous hypothesis (e.g., a life-threatening disease) that has support. That is, the system will search for evidence for and against the more dangerous hypothesis first.

Just as the design of clusters and combining functions give structure to the expert's descriptions of evidential belief, control parameters and control rules organize the expert's strategic knowledge. Control parameters define a space of diagnostic situations, called the *control space*, distinct from the *belief space* of evidential support for hypotheses. In accordance with Principle 1, both the control space and the belief space are constructed from task-level terms. The representation of the control space is designed to facilitate knowledge acquisition from experts rather than forcing them to abide by implementation decisions that they often do not understand. As recommended by Principle 2, MUM selects actions based on declarative control rules; they describe control decisions in terms of explicit control parameters, rather than as unexplainable procedures. In accordance with Principle 3, the design of MUM does not ask the expert to generalize beyond the diagnostic situations with which he or she is familiar.

As combining functions prescribe local combinations of evidence, control rules represent local control decisions. Local control rules have the same relation to global conflict resolution strategies (e.g., "choose the most specific rule") as local combining functions have to their global counterparts (e.g., Bayes' rule). Again the local context facilitates acquisition and makes errors of omission more transparent. When control rules conflict, the cause is missing control knowledge in a particular context. For example, the `prefer-cheap-to-confirming` rule resolves the conflict between more general preference rules, one that says "prefer actions that are potentially confirming" and the other that specifies "prefer actions that cost less." The tradeoff is acquired from a particular diagnostic situation.

9 Implications for the design of architecture support tools

In the previous section we emphasized the design of knowledge *representations* to facilitate knowledge acquisition, but the principles in Section 7 also have practical implications for the design of software support for knowledge *engineering*. Specifically, the principles guide the design of *task-specific architectures*. A task-specific architecture integrates particular knowledge representation formalisms and problem solving strategies to perform a well-defined task, such as hierarchical classification.[10] The point for knowledge acquisition is that task-specific architectures can provide a language of task-level terms to the expert and a way for knowledge engineers to implement these terms

[10]Task-specific architectures have been designed for many familiar tasks. Among them are varieties of classification and diagnosis (Bylander and Mittal, 1986; Clancey, 1985) and design and configuration (Brown 1985; Howe, Dixon, Cohen, and Simmons, 1986; Marcus et al., 1985).

declaratively and at the appropriate level of generality, hiding the implementation.

This section presents a hierarchy of knowledge engineering tools for an architecture called MU that is a generalization of MUM. Figure 5.5 illustrates some of the structure of task-level constructs that MU generalized from MUM; for example, triggering and evidential combination are instances of *inferential relations*, which automatically propagate values through a symbolic inference net.

Figure 5.1 shows the organization of software support for the MU architecture. The three tiers correspond to functional levels. The left column shows the hierarchical relationship among tools. The knowledge acquisition interface is constructed from functionality supplied by the shell, which is built on top of implementation primitives supplied by an AI toolbox. The center column shows the objects that a user would work with at each level; experts would use application-specific terms that are instantiations of task-level constructs, which are in turn implemented using primitives provided by the AI programming environment. The right column lists some of the services provided by software at each level.

At the base of the hierarchy are the implementation level tools. Instead of Lisp, the primitives are AI programming constructs: frames and slots with inheritance and attached procedures, "worlds" for assumption-maintenance, and graphical displays. The software support is standard technology; we currently use the commercial product KEE.[11] The primary user of these tools is the knowledge engineer. Figure 5.6 shows an implementation-level view of part the knowledge base for MUM, reimplemented in MU.

The middle level is the shell – the software that implements a "virtual machine" that operates on task-level constructs. Supporting a virtual machine level is a natural application of Principle 1. The shell is a set of tools, some that support runtime operations, such as propagating the effects of data through an inference net, and others that provide an interface for customizing task-level terms (defined by the architecture) for a specific application. Task-level constructs are implemented as objects using the AI toolbox, but can be viewed by the user as primitives.[12] For instance, one can relate data to hypotheses with an evidential relation or a triggering relation without thinking about how those relations are implemented. Figure 5.7 shows a virtual machine view of part of the evidential support relation for MUM.

The top tier is the knowledge acquisition interface, a set of tools that together present a "user illusion" (Kay, 1984) of a language of application-specific instantiations of constructs provided by the architecture. For example, `classic-angina` is an instantiation of a cluster, and it is presented to the expert as an object related to other clusters and data by links in a graph of evidential support (such as Figure 5.7). The primary function of the knowledge acquisition interface is to make it easy for experts

[11] Which is, of course, a trademark and product of IntelliCorp.

[12] In our implementation, they are represented as class frames, slots, slot facets, attached demons, and method functions.

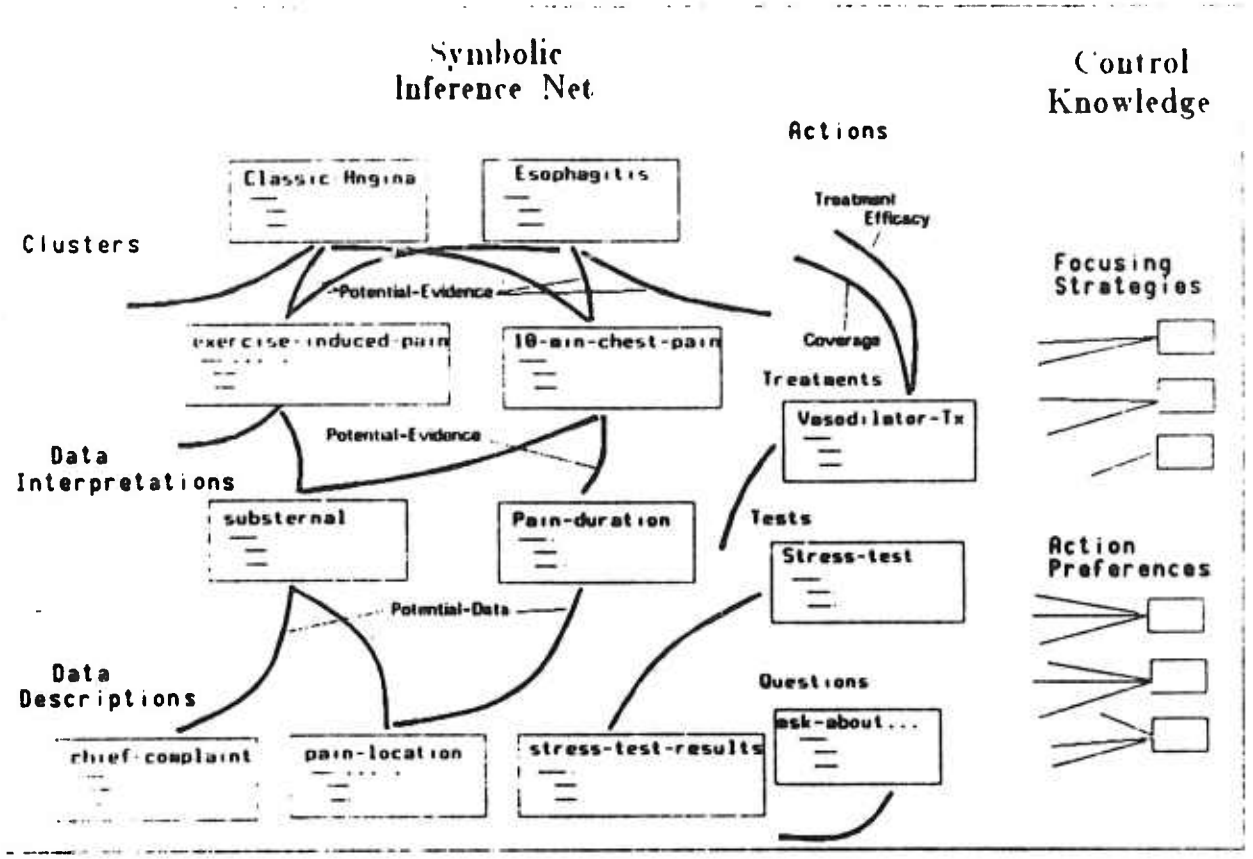


Figure 5.5: The structure of a MU knowledge base.

In the MU architecture, objects in a symbolic inference net are connected by *inferential relations* that propagate symbolic values. For example, the potential-evidence relation propagates *belief states*, such as supported and confirmed. At each node, a local *combining function* determines the belief state of the current node as a function of the belief states of nodes contributing potential evidence. The control knowledge is used to focus (e.g., decide which clusters to concentrate on) and to choose among possible actions (such as prescribing a test), given the state of the objects in the net and characteristics of actions (tests and treatments are actions).

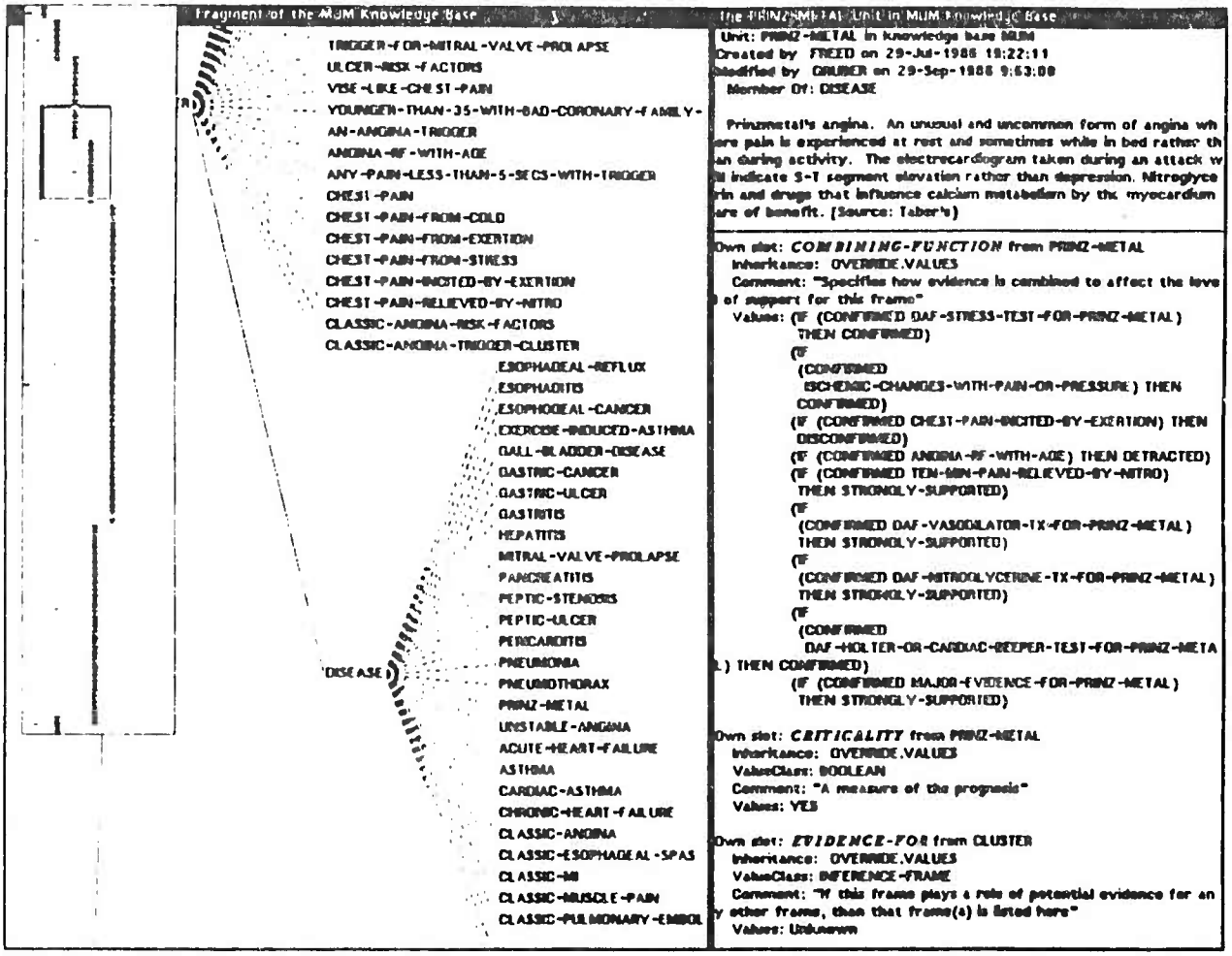


Figure 5.6: The Implementation-level View: A fragment of the MUM knowledge base as displayed by KEE.

The objects in the user's view are implementation-level objects: frames, annotated slots, and inheritance relationships. The graph shows a fragment of a hierarchy of frames. They are organized by their implementation. The window on the left shows some of the clusters and diseases (a subclass of clusters) for MUM. The window on the right shows a KEE display of a disease frame. The semantics of slots are defined by the architecture; for example, all clusters have a slot for combining-function, defined in the `clusters` class frame. `prinz-metal` is a kind of cluster, and it instantiates its own combining function.

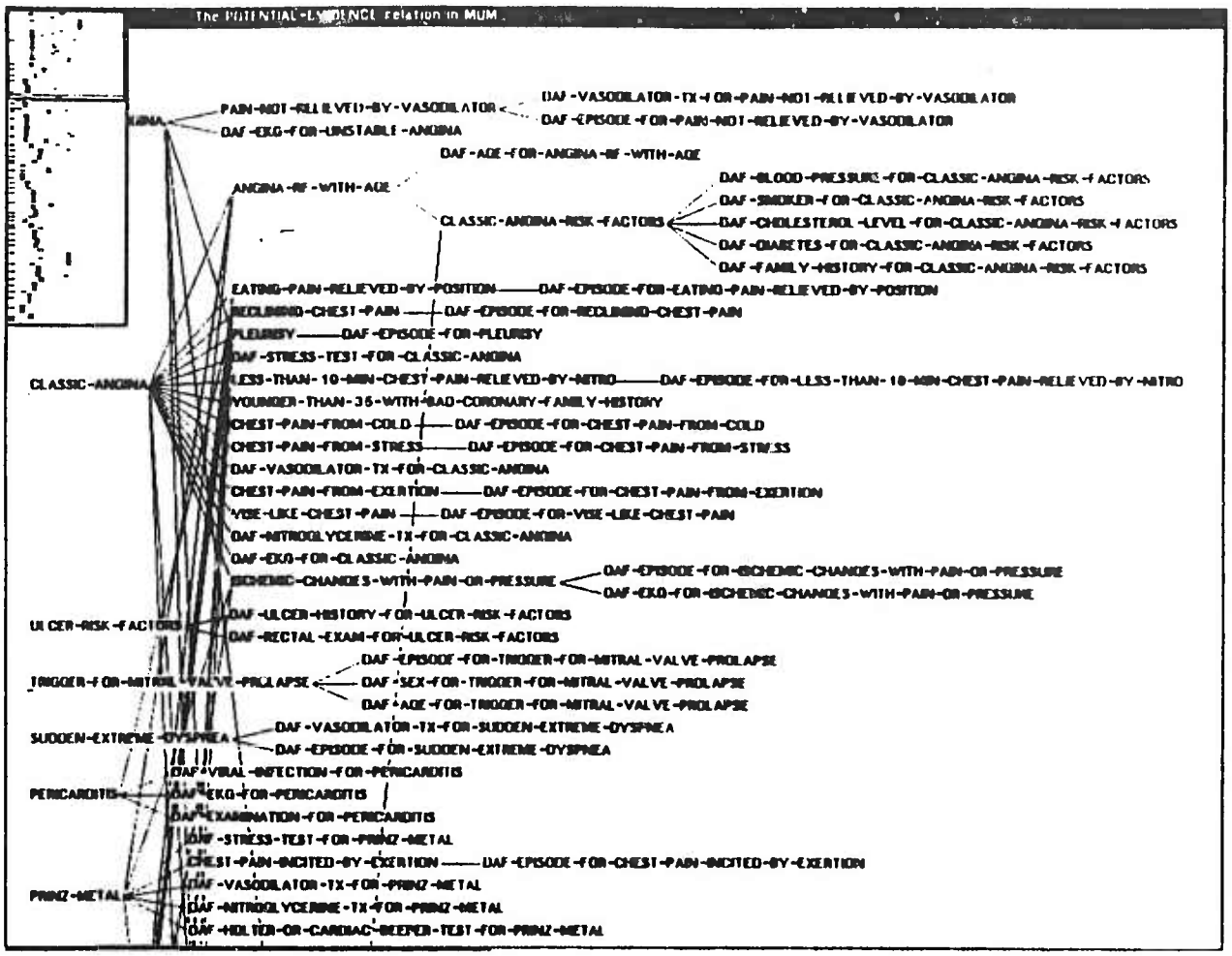


Figure 5.7: The Architectural View: A fragment of the evidential support relation in MUM.

This lattice shows one kind of inferential relation in MUM, the evidential support relation (`potential-evidence`). The nodes in the lattice represent assertions that may be believed. The links represent the inference paths that evidence may take; belief in one node is propagated (i.e., from right to left) to other nodes for which it is evidence (that it may support or detract). The expert or knowledge engineer can select nodes to edit them or add new nodes, and the graph displays the evidential context. Similar graphs are available for other inferential relations provided by the MU architecture, such as triggering and treatment efficacy, and each relation may be viewed in both directions. This view of the knowledge base differs from the frame hierarchy of Figure 5.6 in that the structure represents evidential rather than hierarchical relationships – that is, the structure of the knowledge rather than the implementation.

to formulate their expertise in the available language. A practical effect of applying Principle 1 is that the language is restricted to task-level terms. This allows one to build a knowledge acquisition tool that can apply specific heuristics for acquiring them, as is done in SALT (Marcus, 1987).

When task-level terms are represented declaratively as objects, meta-knowledge (Davis and Buchanan, 1984) about how to acquire task-level terms can be represented as annotations to those objects. This straightforward application of Principle 2 allows one to use simple syntactic techniques to improve the user interface for knowledge acquisition. A surprising amount of leverage can be achieved by using conventional data entry techniques, which we will call *form-filling*, to elicit knowledge from experts. Form filling is a generalization of the "fill in the blank" style of data entry, where each "blank" is labeled and presented in a context. The legal input values are highly constrained and possible values are enumerated when known. On-line help is conveniently accessible, in the form of descriptions of the expected input and examples. For instance, choosing from a menu is a simple kind of form filling (for a single "blank"). A more sophisticated example is the rule editor shown in Figure 5.8, a kind of "language-specific editor" for acquiring rules of various kinds in MU. It uses descriptions of task-level objects to restrict the user's input to semantically valid choices. This technique is similar to the menu-based approach to natural language interfaces described in (Tenant, Ross, Saenz, Thompson, and Miller, 1983). A better example is OPAL (Musen, Fagen, Combs, and Shortliffe, 1987) the knowledge acquisition interface for the ONCOCIN expert system, which uses form-filling to acquire the *majority* of the expert knowledge used in specifying treatment plans for cancer therapy. Form filling is a viable knowledge acquisition technique because the terms that the expert instantiates (e.g., the "blanks" in a form) are explicit and declarative, so that each primitive can be annotated with meta-level descriptions to constrain and validate input. Furthermore, integrating the representations used by knowledge acquisition tools with the shell and the implementation environment is possible because the design of the system anticipated acquisition.

10 Towards Automated Knowledge Acquisition

If the problem of knowledge acquisition is viewed as representational mismatch, the primary contribution of design for acquisition is to make the notation for expressing knowledge more comprehensible and accessible to those with the knowledge. An analysis of successful knowledge acquisition tools (Bennett, 1985; Boose, 1984; Davis, 1976; Eshelman and McDermott, 1987; Kahn, Nowlan, and McDermott, 1984; Marcus et al., 1985) suggests that they satisfy two requirements: to identify the kinds of knowledge to expect from the user, and to provide a functional mapping from user input to implementation primitives. When the underlying architecture supports task-level primitives, the first is accomplished and the second is simplified. Thus design for

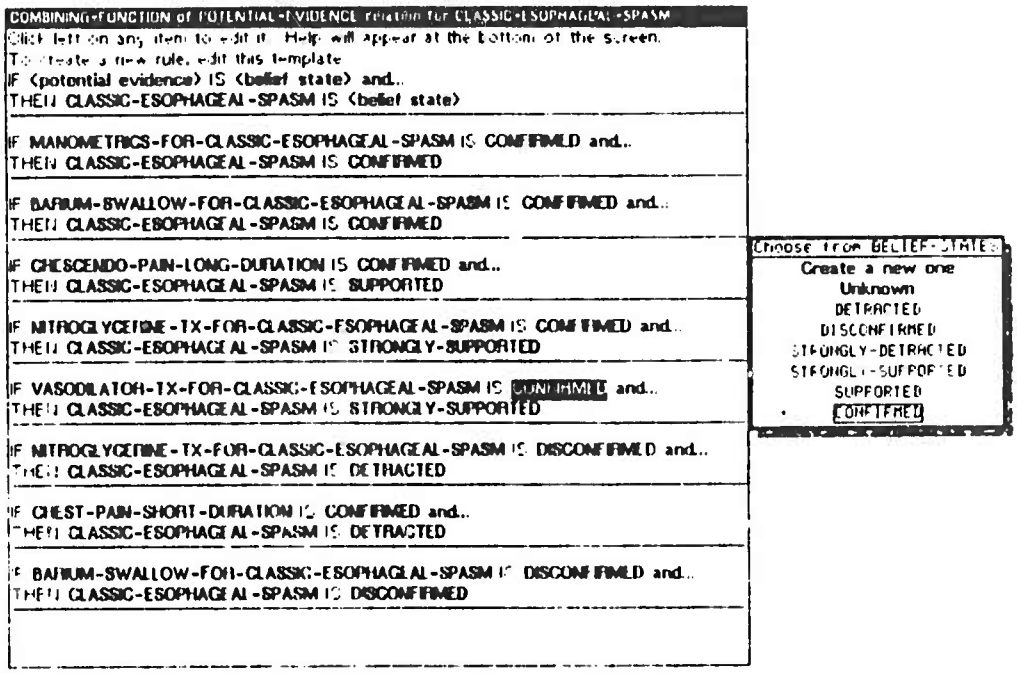


Figure 5.8: The Expert's View: A knowledge-based rule editor for acquiring a combining function.

The rule editor is a sophisticated instance of conventional data entry technology: form-filling. Each term in the rule editor can be selected with a mouse; they are the "blanks" to fill. In the example, the set of rules comprises a symbolic combination function, computing the belief in the diagnosis `classic-esophageal-spasm` as a function of several sources of evidence. The syntax of the rules is supplied as a parameter to the editor, and can be seen in the rule template at the top of the window. In this case, the left hand side of a rule is a statement about belief in one of the clusters which serve as `potential-evidence` for this disease, and the right hand side is always a statement about belief in the diagnosis. The user has selected a belief term from the left hand side of a rule. The meta-level description of combining functions for `potential-evidence` tells the editor that only members of a class called `belief-states` are allowed for this term, and so a menu is presented. If the user chooses to create a new `belief-state`, a form for creating new instances of that class is invoked.

acquisition facilitates knowledge acquisition by both human and machine.

Yet the problem of knowledge acquisition can go beyond representation and implementation issues. It may be that for some kinds of expertise, it is difficult to design *any* notation comprehensible to the expert that can also be executed. If an expert diagnostician is not accustomed to formalizing his or her expertise, there may be no *natural* notation other than the *cases* with which he or she works. For this kind of expertise, induction from examples can be an appropriate acquisition methodology. An inductive learning program can transform knowledge in the form of examples, which alone are inadequate to drive a knowledge system, into more general knowledge of the sort useful to the system.[13]

Acquiring control strategies is an example where a knowledge acquisition methodology can profit from augmenting a good design with induction techniques. Experts who are not familiar with programming may have difficulty writing general control rules, even if they are specified in a comprehensible language of control parameters. Experience with MUM has shown that a good way to acquire these rules is by analyzing physicians' workups on actual patients.[14] This suggests a knowledge acquisition tool that asks about control parameters in the context of decision trees (in the sense of Hannan and Politakis, 1985). Each node in the decision tree corresponds to a decision about what to do next (e.g., test to perform); each arc represents a possible outcome (e.g., test result). The tree contains a wealth of *implicit* control knowledge, in the choice of actions and the order they are prescribed. The role of the acquisition tool is to elicit example decision trees, and to walk the expert through hypothetical cases (paths in the tree) asking for control parameters pertinent to each decision. It could ask questions such as "What factors influenced your decision to do action X instead of Y?" The decision tree is then annotated with these *reasons* for action, and inductive techniques are used to find patterns for generating plausible control rules. The success of the induction still depends, however, on the description language for generalizations (i.e., the *bias* (Utgoff, 1986)). Thus integrating induction with "interviewing" style knowledge acquisition ultimately requires that the proper task-level terms, in this case control parameters, be designed.

11 Conclusions

We described three principles of design for acquisition and demonstrated their application in an architecture where knowledge about evidential combination and knowledge about control can be acquired from an expert. We conclude that proper design

[13] If the system only knew about a set of examples, and had no generalizations, it would be the extreme of "brittleness": it would reduce to a lookup table.

[14] Workups are a natural representation of diagnostic procedure for physicians; they are often published in medical journals. Specific workups for a set of patients can be merged to produce a *workup graph*, which is a compact form of a decision tree.

of knowledge representation primitives can reduce the representational mismatch between implementation-level and task-level formalisms and thereby facilitate knowledge acquisition. We also conclude that emphasizing knowledge *acquisition* in the design of an architecture is consistent with good knowledge *engineering*; if knowledge acquisition tools are designed with the architecture, they can be integrated with runtime and implementation-level software. The function of knowledge acquisition interfaces is made easier when the underlying architecture supports "acquirable" primitives. Finally, we proposed a technique to address a fundamental limitation of the "intelligent interface" approach to automating knowledge acquisition. When the expert cannot formulate the necessary knowledge in any notation, then expert-guided induction may facilitate generalization from examples of problem solving. However, the success of induction still depends on whether the knowledge engineer can devise the proper language of generalizations – the right task-level terms.

12 REFERENCES

Abrett, G. & Burstein, M. The KREME knowledge editing environment. *International Journal of Man-machine Studies*, in press.

Barr, A., & Feigenbaum, E. A. (Eds.) (1981). *The Handbook of Artificial Intelligence*, chapter 5. Volume 1, William Kaufmann, Inc.

Barto, A. G., & Anandan, P. (1984). Pattern Recognizing Stochastic Learning Automata. COINS Technical Report 84-30, University of Massachusetts.

Bennett, J. S. (1986). COAST: A task-specific tool for reasoning about configurations. Technical Report, Teknowledge, Inc., Palo Alto, California.

Bennett, J. S. (1985). ROGET: A knowledge-based consultant for acquiring the conceptual structure of a diagnostic system. *Journal of Automated Reasoning*, **1**(1), pp. 49–74.

Bennetts, R. G. (1984). *Design of Testable Logic Circuits.* Reading, MA: Addison-Wesley.

Berliner, H. (1974). *Chess as problem solving: the development of a tactics analyzer.* Department of Computer Science, Carnegie-Mellon University.

Biggs, S. F., & Mock, T. J. (1983). An Investigation of Auditor Decision Processes in the Evaluation of Internal Controls and Audit Scope Decisions. *Journal of Accounting Research*, Spring 1983.

Bonisonne, P. (1985). Reasoning with uncertainty in expert systems. *International Journal of Man-Machine Studies*, 22:3.

Boose, J. H. (1984). Personal construct theory and the transfer of human expertise. *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX, August, pp. 27–33.

Brown, D. C. (1985). Capturing mechanical design knowledge. *Proceedings of the 1985 International Computers in Engineering Conference*, ASME, Boston, MA, August.

Brown, D. C. & Chandrasekaran, B. (1985). Expert systems for a class of mechanical design activity. In J. Gero (Ed.), *Knowledge Engineering in Computer-aided Design.* Amsterdam: North-Holland.

Buchanan, B. G., Barstow, D. K., Bechtel, R., Bennett, J., Clancey, W., Kulikowski, C., Mitchell, T., & Waterman, D. A. (1983). Constructing an Expert System. In F. Hayes-Roth, D. A. Waterman, & D. B. Lenat (Eds.), *Building Expert Systems*, Reading, MA: Addison-Wesley, 1983.

Buchanan, B. G. & Shortliffe, E. H. (Eds.) (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Reading, MA: Addison-Wesley.

Bylander, T. & Mittal, S. (1986). CSRL: A language for classificatory problem solving and uncertainty handling. *AI Magazine*, **7**(3), pp. 66–77.

Carver, N. F., Lesser, V. R., & McCue, D. L. (1983). Focusing in plan recognition. *Proceedings of the AAAI.*

Chandrasakeran, B. (1986). Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert*, **Fall**, pp. 23–30.

Chandrasakeran, B., Mittal, S., & Smith, J. W. (1982). Reasoning with uncertain knowledge: the MDX approach. *Proceedings of the Congress of American Medical Informatics Association*, San Francisco, pp. 335–339.

Clancey, W. J. (1983). The advantages of abstract control knowledge in expert system design. *Proceedings of the Third National Conference on Artificial Intelligence*, Washington, D. C., August, pp. 74–78.

Clancey, W. J. (1985). Heuristic Classification. *Artificial Intelligence*, **27**, pp. 289–350.

Clancey, W. J. (1984). Classification Problem Solving. *Proceedings of the AAAI*, p. 49.

Clancey, W. J. (1986). From GUIDON to NEOMYCIN and HERACLES in twenty short lessons. *AI Magazine*, **7**(3), pp. 40–60.

Cohen, P. R. (1983). *Heuristic reasoning about uncertainty: An artificial intelligence approach.* Doctoral dissertation, STAN-CS-83-986, Stanford University. Also in press by Pitman Publishers Ltd. London.

Cohen, P. R. (1984). Progress Report on the Theory of Endorsements: A Heuristic Approach to Reasoning About Uncertainty. COINS Technical Report 84-15, University of Massachusetts.

Cohen, P. R. (1986). Managing Uncertainty. Department of Computer and Information Science, University of Massachusetts. Submitted to *Third Conference on Artificial Intelligence Applications*, Orlando, February 1987.

Cohen, P., Day, D., Delisio, J., Greenberg, M., Kjeldsen, R., Suthers, D., & Berman, P. (1987). Management of uncertainty in medicine. *Proceedings of the IEEE Conference on Computers and Communications*, Pheonix, Arizona, February. Also, *International Journal of Automated Reasoning*, Spring 1987.

Cohen, P. R., Davis, A, Day, D., Greenberg, M., Kjeldsen, R., Lander, S., & Loiselle, C. (1985). Representativeness and uncertainty in classification systems. *AI Magazine*, Fall 1985.

Cohen, P. R., & Feigenbaum, E. (1982). *The Handbook of Artificial Intelligence, Volume III.* Los Altos, CA: William F. Kaufmann, Inc.

Cohen, P. R., Greenberg, M., & Delisio, J. (1987). MU: A development environment for prospective reasoning systems. *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, Washington, July 1987, forthcoming.

Cohen, P. R., & Grinberg, M. R. (1983). A theory of heuristic reasoning about uncertainty. *The AI Magazine*, Summer 1983.

Cohen, P. R. & Gruber, T. R. (1985). Reasoning about uncertainty: A knowledge representation perspective. *Pergamon Infotech State of the Art Report.* Also COINS Technical Report 85–24, Department of Computer and Information Science, University of Massachusetts.

Cohen, P. R., & Kjeldsen, R. (1987). Information Retrieval by Constrained Spreading Activation in Semantic Networks. *Information Processing and Management.*

Cohen, P. R., & Lieberman, M. D. (1983). FOLIO: An expert assistant for portfolio managers. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 212-215.

Cohen, P. R., Shafer, G., & Shenoy, P. (1987). Modifiable Combining Functions. EKSL Report 87-05, Department of Computer and Information Science, University of Massachusetts.

Cohen, P. R., & Stanhope, P. (1986). Finding research funds with the GRANT system. *Proceedings of the Sixth International Workshop on Expert Systems and Their Applications*, April 28-30, 1986, Avignon, France.

Collins, A. (1978 (a)). Fragments of a theory of human plausible reasoning. *Theoretical Issues in Natural Language Processing.* D. Waltz, Ed. Urbana, IL: University of Illinois.

Collins, A. (1978 (b)). *Human Plausible Reasoning.* Cambridge, MA: Bolt, Beranek and Newman, Inc. Report No. 3810.

Collins, A., Warnock, E., Aiello N., & Miller, M. (1975). Reasoning from incomplete knowledge. In *Representation and Understanding*, D.G. Bobrow and A. Collins (Eds.), New York: Academic Press.

Coombs, C. H., Dawes, R. M., & Tversky, A. (1970). *Mathematical Psychology.* Englewood Cliffs, NJ: Prentice Hall, Inc.

Davis, R. (1985). Interactive transfer of expertise. In *Rule-based expert systems*, B. Buchanan and E. Shortliffe, (Eds). Addison-Wesley Publishing Company.

Davis, R. (1976). Applications of meta-level knowledge to the construction, maintenance, and use of large knowledge bases. Doctoral dissertation, Computer Science Department, Stanford University. Reprinted in R. Davis & D. B. Lenat (Eds.), *Knowledge-Based Systems in Artificial Intelligence*, New York: McGraw-Hill, 1982.

Davis, R. & Buchanan, B. G. (1984). Meta-level knowledge. In B. G. Buchanan & E. H. Shortliffe (Eds.), *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Reading, MA: Addison-Wesley, 1984.

de Kleer, J., Doyle, J., Steele, G. L., Jr., & Sussmann, G. J. (1977). AMORD: Explicit Control of Reasoning. *Proceedings of the Symposium on Artificial Intelligence and Programming Languages*, SIGPLAN Notices 12(8), and SIGART Newsletter, 64, pp. 116-125.

Dempster, A. P. A generalization of Bayesian inference. *Journal of the Royal Statistical Society*, series B, 30.

Dietterich, T. G. (1982). Learning. In P. Cohen and E. Feigenbaum (Eds.) *The Handbook of Artificial Intelligence, Volume III.* Los Altos, CA: William Kaufmann, Inc.

Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence*, 13.

Doyle, J. (1983 (a)). The ins and outs of reason maintenance. *Proceedings of the International Joint Conference on Artificial Intelligence,* 1983.

Doyle, J. (1983 (b)). *Some theories of reasoned assumptions: Some essays in rational psychology.* CMU-CS-83-125. Carnegie-Mellon University.

Duda, R. O., Hart, P. E., & Nilsson, N. J. (1976). Subjective Bayesian methods for rule-based inference systems. *Proceedings of the 1976 National Computer Conference.*

Erman, L. D., Hayes-Roth, F., Lesser, V. R., & Reddy, D. R. (1980). The HEARSAY-II speech understanding system: integrating knowledge to resolve uncertainty. *Computing Surveys*, 12.

Eshelman, L. & McDermott, J. (1987). MOLE: A tenacious knowledge acquisition tool. *International Journal of Man-machine Studies.*

Eshelman, L. & McDermott, J. (1986). MOLE: A knowledge acquisition tool that uses its head. *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, pp. 950-955.

Fikes, R., Hart, P., & Nilsson, N. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4), pp. 251-288.

Ford, F. N. (1985). Decision support systems and expert systems: a comparison. *Journal of Information and Management*, Vol. 8, pp. 21-26.

Ginsberg, M. L. (1984). Non-monotonic reasoning using Dempster's Rule. *Proceedings AAAI-84.*

Gruber, T. R., & Cohen, P. R. (1987). Design for acquisition: principles of knowledge system design to facilitate knowledge acquisition. To appear in the *International Journal of Man Machine Studies.*

Gruber, T. R., & Cohen, P. R. (1987). Principles of Design for Knowledge Acquisition. *Proceedings of the Third IEEE Artificial Intelligence Applications Conference*, Orlando, Florida, February 1987.

Hannan, J. & Politakis, P. (1985). ESSA: An approach to acquiring decision rules for diagnostic expert systems. *Proceedings of the Second Conference on Artificial Intelligence Applications*, pp. 520–525.

Hayes-Roth, B., Garvey, A., Johnson, M. V., & Hewett, M. (1986). A layered environment for reasoning about action. KSL Report No. 86-38, Department of Computer Science, Stanford University.

Hayes-Roth, F. (1978). The role of partial and best matches in knowledge systems. *Pattern Directed Inference Systems*, D. Waterman, D. Hayes-Roth, and D. Lenat (Eds). Academic Press.

Hayes-Roth, F. (1985). A blackboard architecture for control. *Artificial Intelligence*, Vol. 26, pp. 251-321.

Hayes-Roth, F, & Lesser, V. (1977). Focus of attention in the HEARSAY-II speech understanding system. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence.*

Henrion, M. & Cooley, D. R. (1987). An experimental comparison of knowledge engineering for expert systems and for decision analysis. In *Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI.*

Holland, J. H. Escaping Brittleness: The Possibilities of General Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. University of Michigan.

Howard, R. A. (1966). Decision Analysis: Applied Decision Theory. In D.B Hertz and J. Melese, (Eds). *Proceedings of the Fourth International Conference on Operational Research*, pp. 55-71. Wiley, New York.

Howard, R. A. (1968). The foundations of decision analysis. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4, pp. 211-219.

Howe, A. (1986). *A Constructive Approach to Decision Making.* Master's thesis, Department of Computer and Information Science, University of Massachusetts.

Howe, A., & Cohen, P. R. (1986). Comparing Alternatives in Decision Making. EKSL Memo, University of Massachusetts.

Howe, A. E., Dixon, J. R., Cohen, P. R., & Simmons, M. K. (1986). DOMINIC: A domain-independent program for mechanical engineering design. *International Journal for Artificial Intelligence in Engineering*, 1(1), July, pp. 23–29.

Hwang, C. L., & Yoon, K. (1981). *Multiple Attribute Decision Making: Methods and Applications.* Springer-Verlag, Berlin.

Kahn, G., Breaux, E. H., Joseph, R. L., & DeKlerk, P. (1987). An intelligent mixed-initiative workbench for knowledge acquisition. *International Journal of Man-machine Studies*, in press.

Kahn, G., Nowlan, S. & McDermott, J. (1984). A foundation for knowledge acquisition. *Proceedings of the IEEE Workshop on Principles of Knowledge-base Systems*, Denver, Colorado, December, pp. 89–98.

Kahn, G., Nowlan, S. & McDermott, J. (1985). MORE: an intelligent knowledge acquisition tool. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, pp. 581–584.

Kahneman, D., Slovic, P., & Tversky, A. (1982). *Judgment under uncertainty: Heuristics and biases.* Cambridge University Press.

Kahneman, D., & Tversky, A. (1982). Subjective probability: A judgment of representativeness. In D. Kahneman, P. Slovic, and A. Tversky, *Judgment under uncertainty: Heuristics and biases.* Cambridge University Press, p. 35.

Kay, A. (1984). Computer Software. *Scientific American*, **251**(3), September, pp. 52–59.

Kim, J. M. & Pearl, J. (1983). A computational model for causal and diagnostic reasoning in inference systems. *Proceedings IJCAI-83.*

Kingsland, III, L. C. & Lindberg, D. A. B. (1986). The criteria form of knowledge representation in medical artificial intelligence. *Proceedings of the Fifth Conference on Medical Informatics*, Washington, D. C., October 26–30, pp. 12–16.

Kjeldsen, R., & Cohen, P. R. (1987). The Evolution and Performance of the GRANT System. *IEEE Expert*, Spring 1987.

Leal, A. & Pearl, J. (1977). An interactive program for conversational elicitation of decision structures. *IEEE Transactions on Systems, Man and Cybernetics*, 7(5).

Lenat, D. B. (1976). *AM: an artificial intelligence approach to discovery in mathematics as heuristic search.* (Doctoral dissertation), Department of Computer Science report STAN-CS-76-570, Stanford University.

Lenat, D. B. & Brown, J. S. (1983). Why AM and Eurisko appear to work. *Proceedings AAAI-83.*

Lenat, D, Prakash, M., & Shepherd, M. (1986). CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks. *AI Magazine*, 6(4), pp. 65-85.

Levesque, H. J. (1984). A logic of implicit and explicit belief. *Proceedings AAAI-84.*

London, P. E. (1978). *Dependency networks as a representation for modeling in general problem solver.* (Doctoral dissertation), Department of Computer Science Technical Report 698, University of Maryland.

Lowrance, J. P., & Garvey, T. P. (1982). Evidential reasoning: a developing concept. *Proceedings of the International Conference on Cybernetics and Society.*

Luce, R. & Raiffa, H. (1957). *Games and Decisions.* Wiley and Sons, Inc., New York.

Marcus, S., Caplain, G., McDermott, J., & Stout, J. C. (1986). Making SALT Generic. Department of Computer Science, Carnegie-Mellon University.

Marcus, S., McDermott, J., & Wang, T. (1985). Knowledge acquisition for constructive systems. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, pp. 637–639.

Marcus, S. (1987). Taking backtracking with a grain of SALT. *International Journal of Man-machine Studies.*

McCarthy, J. (1958). Mechanization of Thought Processes. *Proceedings of the Symposium, National Physics Laboratory*, 1, pp. 77-84, London.

McCarthy, J. (1968). Programs with Common Sense. *Semantic Information Processing*, pp. 403-418. M. Minsky, Ed. Cambridge, MA: The MIT Press.

McDermott, J. (1983). Extracting knowledge from expert systems. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, August, pp. 100–107.

McDermott, D. & Doyle, J. Non-monotonic Logic I. *Artificial Intelligence*, 13, pp. 27-39.

Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (1983). *Machine Learning: An Artificial Intelligence Approach.* Palo Alto, CA: Tioga.

Mostow, D. J. (1983). Machine transformation of advice into a heuristic search procedure. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach.* Palo Alto: Tioga, 1983, pp. 243–306.

Musen, M. A, Fagen, L.M., Combs, D. M., & Shortliffe, E. H. (1987). Using a domain model to drive an interactive knowledge editing tool. *International Journal of Man Machine Studies.*

Neches, R., Swartout, W. R., & Moore, J. (1985). Enhanced maintenance and explanation of expert systems through explicit models of their development. *Transactions on Software Engineering*, **SE-11**(11), pp. 1337–1351.

Neches, R., Swartout, W. R., & Moore, J. (1984). Explainable (and maintainable) expert systems. *Proceedings of the IEEE Workshop on Principles of Knowledge-base Systems*, Denver, Colorado, pp. 173-184.

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, Vol. 18, pp. 87-127.

Nii, H. P. & Feigenbaum, E. A. (1977). Rule-based understanding of signals. Memo STAN-CS-77-612, Stanford University.

Nilsson, N. (1980). *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga.

Nilsson, N. (1984). Probabilistic Logic. SRI AI Center Technical Note 321, SRI International, Menlo Park, CA.

North, D. W. (1968). A tutorial introduction to decision theory. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(4).

Patel, V., & Groen, G. (1986). Knowledge based solution strategies in medical reasoning. *Cognitive Science*, Vol. 10, pp. 91-116.

Pauker, S. G., Gorry, A., Kassirer, J. P., & Schwartz, W. B. (1976). Towards the simulation of clinical cognition: taking a present illness by computer. *American Journal of Medicine*, 60, pp. 981–996.

Payne, J. (1982). Contingent decision behavior. *Psychological Bulletin*, 92(2), pp. 382-402.

Payne, J, Braunstein, M., & Carroll, J. (1978). Exploring Predecisional Behavior: An Alternative Approach to Decision Research. *Organizational Behavior and Human Performance*, Vol. 22, pp. 17-44.

Pearl, J. (1982). Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach. *Proceedings of the National Conference on Artificial Intelligence*, Pittsburgh, PA, pp. 133-136.

Pearl, J., Leal, A., & Saleh, J. (1982). GODDESS: A Goal-Directed Decision Structuring System. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 4, pp. 250-262.

Pople, H. (1977). The formation of composite hypotheses in diagnostic problem solving — an exercise in synthetic reasoning. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, Massachusetts, pp. 1030–1037.

Raiffa, H. (1970). *Decision Analysis: Introductory Lectures on Choices Under Uncertainty.* Reading, MA: Addison-Wesley.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, 13.

Rich, E. (1983). Default reasoning as likelihood reasoning. *Proceedings AAAI-83.*

Sacerdoti, E. D. (1975). *A structure for plans and behavior.* New York: American Elsevier.

Sacerdoti, E. D. (1979). Problem solving tactics. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pp. 1077-1085.

Sage, A. P. & Rouse, W. B. (1986). Aiding the human decisionmaker through the knowledge based sciences. *IEEE Transactions on Systems, Man and Cybernetics*, 16(4), pp. 511-522.

Sage, A., & White, C. (1984). ARIADNE: A Knowledge-Based Interactive System for Planning and Decision Support. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 14, pp. 35-47.

Salmon, W. C. (1967). *The Foundations of Scientific Inference.* Pittsburgh, University of Pittsburgh Press.

Shafer, G. (1976). *A Mathematical Model of Evidence.* Princeton: Princeton University Press.

Shortliffe, E. (1976). *Computer-Based Medical Consultations:* MYCIN. New York: American Elsevier.

Shortliffe, E. H. & Buchanan, B. G. (1975). A model of inexact reasoning in medicine. *Mathematical Biosciences*, **23**, pp. 351–379.

Simon, H. A. (1981). *The Sciences of the Artificial.* Cambridge, MA: The MIT Press.

Stallman, R. M. & Sussman, G. J. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9.

Stanhope, P. (1986). An Overview of GRANT. 1986 EKSL Memo No. 11. Experimental Knowledge Systems Laboratory, University of Massachusetts, April 1986.

Stefik, M. J. (1980). *Planning with constraints.* (Doctoral dissertation), Department of Computer Science Report 80-784, Stanford University.

Strat, T. M. (1984). Continuous belief functions for evidential reasoning. *Proceedings AAAI-84.*

Svenson, O. (1979). Process Descriptions of Decision Making. *Organizational Behavior and Human Performance*, Vol. 23, pp. 86-112.

Swartout, W. (1983). XPLAIN: A system for creating and explaining expert consulting systems. *Artificial Intelligence*, 21(3), pp. 285-325.

Szolovits, P. & Pauker, S. G. (1978). Categorical and probabilistic reasoning in medical diagnosis. *Artificial Intelligence*, **11**, pp. 115–144.

Tenant, H. R., Ross, K. M., Saenz, R. M., Thompson, C. W., & Miller, J. R. (1983). Menu-based natural language understanding. *Proceedings of the Association for Computational Linguistics*, Massachusetts Institute of Technology, June, pp. 151–158.

Turner, R. (1984). *Logics for Artificial Intelligence.* Chichester: Ellis Howard Limited.

Tversky, A. (1977). Features of Similarity. *Psychological Review*, Vol. 84, No. 4, July 1977.

Tversky, A., & Kahneman, D. (1982). Judgment under uncertainty: Heuristics and biases. In D. Kahneman, P. Slovic, and A. Tversky, *Judgment under uncertainty: Heuristics and biases.* Cambridge University Press, p. 4.

Utgoff, P. (1986). Shift in Bias for inductive concept learning. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach,* Vol. II, Los Altos, California: Morgan Kaufmann.

van Melle, W. (1979). A domain independent production rule system for consultation programs. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence,* August, pp. 923–925.

von Holstein, S. (1971). A Tutorial on Decision Analysis. In *Third Research Conference on Subjective Probability, Utility, and Decision Making,* London.

Wang, M. S., & Courtney, J. F, Jr. (1984). A conceptual architecture for generalized decision support system software. *IEEE Transactions on Systems, Man and Cybernetics,* 14(5).

Weiss, S., Kulikowski, C, & Safir, A. (1977). A model-based consultation system for the long-term management of glaucoma. *Proceedings IJCAI* 5, pp. 826-832.

Wesley, L. P. (1983). Reasoning about control: the investigation of an evidential approach. *Proceedings IJCAI-83,* pp. 203-206.

Winston, P. H. (Ed.). (1975). *The Psychology of Computer Vision.* New York: McGraw Hill.

Zadeh, L. A. (1975). Fuzzy logic and approximate reasoning. *Synthese,* **30**, pp. 407–428.

DISTRIBUTION LIST

addresses	number of copies
Chuian-Chuian Hwong RADC/COES	25
RADC/DOVL GRIFFISS AFB NY 13441	1
RADC/DAP GRIFFISS AFB NY 13441	2
ADMINISTRATOR DEF TECH INF CTR ATTN: DTIC-DDA CAMERON STA BG 5 ALEXANDRIA VA 22304-6145	12
AFIT/LDEE - TECHNICAL LIBRARY BUILDING 640, AREA B WRIGHT-PATTERSON AFB OH 45433-6583	1
AFWAL/FIES/SURVIAC WRIGHT-PATTERSON AFB OH 45433	1
AFHRL/LRS-TDC WRIGHT-PATTERSON AFB OH 45433-6503	1

COMMANDER
NAVAL OCEAN SYSTEMS CENTER
ATTN: TECHNICAL LIBRARY, CODE 9642
SAN DIEGO CA 92152-5000 — 1

UNIVERSITY OF MASSACHUSETTS
P.O. BOX 571
AMHERST MA 01003 — 5

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY
1400 WILSON BOULEVARD
ARLINGTON VA 22209 — 1

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.